My project: Handwriting recognition research project

Contents page:

My project: Handwriting recognition research project

Section 1) Analysis

Section 1.1) Problem Identification:

As a Mathematics and Computer Science student, I understand the importance of handwritten notes for subject assignments and coursework. Taking notes by hand is very instinctive to many students and in some cases much more convenient than typing because drawing diagrams, arrows and certain symbols is crucial to note taking and therefore it makes handwritten note taking much easier and more straightforward than using a computer. Modern technology such as using touchscreens and styluses have lessened the gap between paper and digital note taking, however you need hardware capabilities in your computers which cannot be afforded by everyone.

On the other hand, in terms of organization of notes and security, digital storage and retrieval is a better implementation than handwriting. For example, you can store collections of notes on a cloud network or a portable medium such as USB-drive. This makes notes very accessible because it can be retrieved anytime and there will always be a backup for it.

There are some very fundamental problems with handwriting. Handwritten notes can be messy and disarranged because students can rush and make notes hastily, which affects the organization and therefore the quality of notes being taken. This can be a major hinderance if these notes are intended to be sent to a teacher/professor or is needed for future revision (it might be hard to look back at messy notes). Another issue with handwritten note taking is that they can be easily displaced and lost, because there is (usually) no external backup of these documents. If these handwritten notes are lost, assignments and revision that depend on it, can be substantially delayed.

This is the foundation for my project where I am planning to make a handwriting recognition interface where the user can input handwritten notes, and (using machine learning algorithms) my program can translate and output a reasonable digital version of that text sample.

Why my problem amenable to a computational solution (justification):

The core aspect of my project is letter recognition. I need to make a software that can detect individual letters from any given handwritten templates, recognise which letter in the alphabet it is, and then output them. This is only amenable to a computational solution because there are computational software that specialise in detection and therefore can be used to aid in the solution of my problem. Also, machine learning is a branch of computer science that allows a program to “recognise” and “learn” inputs on the basis of test data and training data. I plan to incorporate machine learning into my project so that my program can efficiently recognise inputs. I can add more test data or amend the working code for machine learning to increase efficiency of letter recognition.

Therefore, my problem is only suited to a computational solution and would not work otherwise, because the core and foundational part of my problem (recognition of letters) requires a branch of computing that specialises in recognition and output of user data.

Section 1.2) Stakeholders for the project:

My project: Handwriting recognition research project

The demographic of the clients that will use my product will mainly consist of students (15-25 year olds) because researching and taking notes is at the centre of their education and learning, and I know they would benefit from a program that will help better organise their notes and will make storing notes more convenient and easier.

However, this product isn't just limited or confined to use for students because people of all ages can write and therefore may require this program to store their work in a more portable fashion. For example, people may have made handwritten portfolios or CVs and might use this program to digitally store and perhaps send their work to employers/businesses.

In this project, I will mainly focus on the recognition of handwritten letters and numbers because letters and numbers are what most handwritten inputs consist of and they need to be recognised well, in order to find a feasible solution for my project. I also think this is the hardest aspect of the whole project due to the high level of complexities there are in recognising different variations of writing letters such as cursive or block script writing. Also, certain letters and numbers appear too similar to each other such as "9", "q".

Interview Questions:

For this project, I have 3 stakeholders. Both stakeholders are mathematicians (one of them is a Maths teacher, and the other one is a dedicated maths student). I wanted to ask them questions regarding their expectations for this project and how I can make their lives as mathematician easier

Q1) First of all, what kind of note taker are you (are your notes neat and tidy or messy!)?

Q2) What (in your opinion) is the advantages and (most importantly) disadvantages of using paper for making notes for mathematics?

Q3) Do you think storing notes in a digital format would be convenient for you (for example for archiving purposes)?

Q4) Do you think you will use my project (assuming it works as intended) for everyday use in teaching/homework submission?

Q5) Thinking from the developer's perspective, what challenges do you think will be encountered in this project (and do you have any suggestions to solve them)?

Q6) What kind of features do you want to see in this project (in terms of GUI layout, features, etc)?

Q7) What are the most important mathematical symbols and expressions?

Q8) What is more important to you, usability of an app or the how feature-rich it is?

I asked 2 of my stakeholders, Mehitabel and Jago (both of them are maths students who fall in my client demographic).

These are Mehitabel's response:

Q1) First of all, what kind of note taker are you (are your notes neat and tidy or messy!)?

I would say that my notes are quite tidy

My project: Handwriting recognition research project

Q2) What (in your opinion) is the advantages and (most importantly) disadvantages of using paper for making notes for mathematics?

An advantage of taking notes for mathematics is that you can clearly set out the steps you are taking to solve and equation so it would allow you to get the correct answer. The disadvantage of using paper for taking mathematics notes is that It can waste a lot as paper as you are working out a question also you might lose the notes that you have made.

Q3) Do you think storing notes in a digital format would be convenient for you (for example for archiving purposes)?

I think storing notes in a digital format would be good for me because the notes are easy to find and go back to if you want to revise it.

Q4) Do you think you will use my project (assuming it works as intended) for everyday use in teaching/homework submission?

I think that I would use your project because it is an efficient algorithm that solves the problem of notes getting lost.

Q5) Thinking from the developer's perspective, what challenges do you think will be encountered in this project (and do you have any suggestions to solve them)?

A challenge that you might encounter in this project is that many peoples handwriting is different so it might be hard for an algorithm to recognise it A suggestion that you could do to help this is to train the algorithm with different types of test data.

Q6) What kind of features do you want to see in this project (in terms of GUI layout, features, etc)?

A feature that would like to see in this project is a graphical interface that is easy to use with big buttons so I know what to do in the program.

Q7) What are the most important mathematical symbols and expressions?

I think the most mathematical symbols and expressions are , an equal's sign pi, addition sign, multiplication sign, division sign and subtraction sign,

Q8) What is more important to you, usability of an app or the how feature-rich it is?

I would say that the usability of an app is important to me because if you have many features it is good but you may not know how to use them well if it is not as obvious.

Jago's response:

Q1) First of all, what kind of note taker are you (are your notes neat and tidy or messy!)?

My notes are somewhat organised, as I have separate notebooks for all my subjects, but they are still full of scribbling out, or trying to fit as much text as possible on one page to save paper, so I am not particularly focused on aesthetics if all the key information is written down somewhere.

Q2) What (in your opinion) is the advantages and (most importantly) disadvantages of using paper for making notes for mathematics?

My project: Handwriting recognition research project

Advantages of paper - easy to draw symbols / equations, quicker for small questions than using software, never runs out of battery life

Disadvantages of paper - hard to keep organised, difficult to cross out / erase mistakes, annoying and slow to keep on copying out lines of work when dealing with long questions, hard to draw accurate diagrams

Q3) Do you think storing notes in a digital format would be convenient for you (for example for archiving purposes)?

I have lots of mathematics notebooks which take up a lot of space in my bedroom at home, so having a digital storage format would save me lots of space and be very convenient for retrieval

Q4) Do you think you will use my project (assuming it works as intended) for everyday use in teaching/homework submission?

The project would be most useful if it could recognise a wider range of mathematical notation, such as fractions, summations, algebra, etc, but for the initial version with letter / number recognition functionality, I would definitely use it for storing hand-written notes and basic calculations, assuming that the process of inputting a document into the program was simple and quick

Q5) Thinking from the developer's perspective, what challenges do you think will be encountered in this project (and do you have any suggestions to solve them)?

I think the challenges in a project like this would include:

- Finding and manipulating a suitably large data set
- Working out how to respond to invalid symbols (if possible)
- Working out how to split up words into their individual characters, and ensure each character image has an appropriate resolution to function within the program

Q6) What kind of features do you want to see in this project (in terms of GUI layout, features, etc)?

I would most like:

- Interface with the original image, showing one-to-one correspondence between a symbol I entered and the character it was recognised as
- A final complete document with the typed-out, formatted text / symbols
- An option to download the document

Q7) What are the most important mathematical symbols and expressions?

Addition (+), subtraction (-), division (÷ or /), multiplication (x or *) and equality (=); fractions, variable names (usually English alphabet letters), symbols used in proofs (⇒ and ∴)

Q8) What is more important to you, usability of an app or the how feature-rich it is?

I would prefer an app with a small number of reliable features, rather than an app where some features work and some features are not as effective

My project: Handwriting recognition research project

My takeaway from stakeholder questions:

The takeaway from this questionnaire is that a handwriting recognition program would be helpful for students as it helps solves many problems such as digital storage (meaning there will always be one backup of your notes) and tidying up notes (writing in digital format is better for people with messy handwriting).

Section 1.3) Problem Decomposition (pre-research):

In this project, the core issues I will have to solve are:

1. Making sure input image is readable for the program
    - Reduce blur and noise in the image by applying image thresholding algorithm (e.g. binary thresholding)
    - The background of the image should be white (so the foreground and letters are clear and visible for recognition)
2. Making sure each character (in the input image) is cropped and recognised individually
    - Recognise and save every character in sequence of human reading (e.g. recognsing each row one at a time starting from the left at each row, etc)
3. Output the results in a readable document
    - Use an efficient machine learning algorithm that has the best accuracy in terms of recognision
    - The process should not be time consuming for the user
4. Make sure the user interface is professional and usable for the user
    - Check all features in the interface work (i.e., no errors when operating)
    - The user experience should be straightforward, and tutorials should be in place for the users to use

Justification for problem decomposition:

For handwritten text to be recognised, I will need to detect, isolate, and perform my recognition algorithm on individual letter in the input image. Only then will I get letter outputs that I can combine with other image data (such as coordinates of letters) to output a text file to the user.

The first step of my handwritten text recognition will be to clean and denoise the input image so that detection of letters and recognition will be easier (as noise can increase the chance of invalid recognition output). Therefore, this is the first bullet point in my decomposition.

The next step of my handwritten text recognition will be detecting letters and cropping them so that the image can be feed in the recognition algorithm so that it can be recognised. This step is crucial in my program because this is where my neural network input is prepared (on which the output to the user depends on) so I need to formulate an algorithm that will prepare the neural network inputs properly.

This is probably the most important part of my final solution, which is making a neural network and giving an output to the user (in a text document format containing words based on handwritten image data). The efficacy and reliability of my program directly depends on the efficacy of my neural network (recognition algorithm), therefore I will need to research neural networks thoroughly, and make an efficient algorithm.

My project: Handwriting recognition research project

The final part of my program solution is making a user interface. This is the final step in my program, because an interface allows the user to communicate with my neural network, so I cannot start my interface development without a working prototype of my neural network.

Section 1.4) Research

For handwriting recognition, I will require knowledge and understanding of machine learning and neural networks.

Section 1.4.1) Machine Learning Research

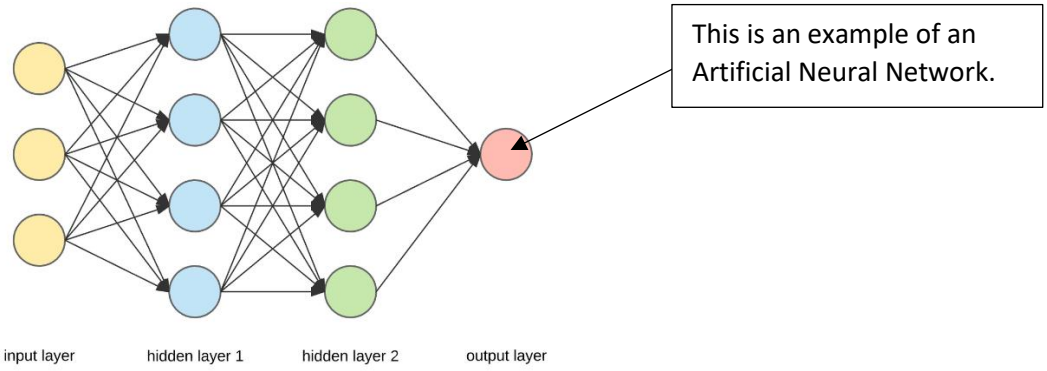
Machine Learning (ML) is method of data analysis using computational algorithms that creates and continuously improves a model that can identify patterns by itself and therefore make decisions (Machine learning - Wikipedia). These models often include complex neural networks, nodes and layers.

In this project, machine learning will play an integral role in recognising and hence outputting text based on image inputs.

Neural Networks

Artificial Neural Networks (ANNs) are networks that are used heavily in processing data during machine learning. These systems are influenced from biological neural networks (like the Human brain) which rely heavily on neurons. All neural networks contain these underlying parts (What are Neural Networks? | IBM):

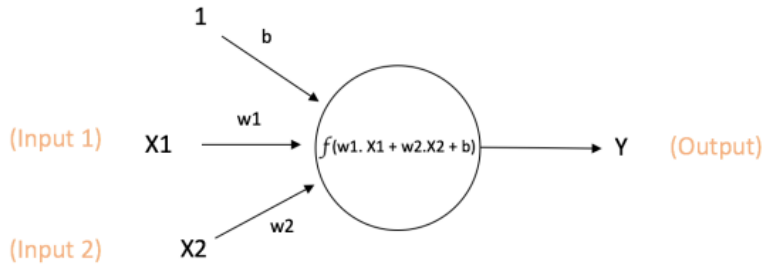
- Nodes(Neurons) – Nodes (also known as neurons in an ANN) are a unit of data that carry a certain activation value based on which processing can be done (in layer functions)
- Input/Output – Input nodes are part of the Input layer, and these nodes are the foundations of the whole ANN. The output layer contains nodes that are a direct result of calculations done using the input layer.
- Layer – Layer is a general term that refers to a group of nodes that have a specific function. An example is an Input layer that contains nodes that have input data.



Deeper look at a single neuron

A neuron (node) is the basic unit of a neural network. The value of each neuron is calculated using the Activation function that uses weighted sums as a parameter. The

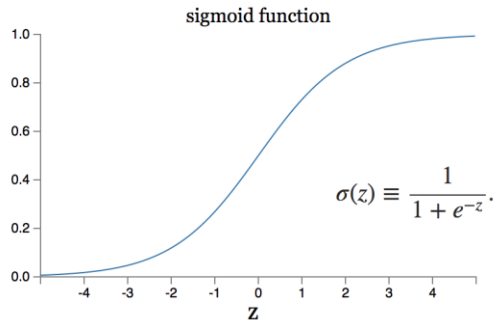
weighted sum ∑(w1a1 + w2a2 +…) of a neuron is the sum of the weights of the arcs that connect the node. This can be seen in the diagram below (A Quick Introduction to Neural Networks – the data science blog (ujjwalkarn.me)):



Output of neuron = Y= f(w1. X1 + w2.X2 + b)

The Activation function has a specific purpose. It is a non-linear function (Activation function - Wikipedia) that is responsible for introducing non-linearity to the output value of the neuron. This is because, to model real life situations, you would need to model the neurons in a way so that they represent the aspect of real life accurately (which has to be non-linear). The activation function takes the weighted sum as an input and uses it in a mathematical function such as the Sigmoid function. There can be other mathematical functions used as well such as ReLU but it depends on the respective neural network (3Blue1Brown - Gradient descent, how neural networks learn). The sigmoid function is the most recognised and excessively used function in Neural networks. (especially for beginner projects).

The sigmoid function is a non-linear function with equation $f(x) = \frac{1}{1+e^{-(x)}}$ and it outputs a non-linear value between 0 and 1 for the value(s) of the weighted sum of any given node.



As you can see for value on the x-axis, this function outputs a value for y-axis that is between 0 and 1.

What is Bias and why is it important in Machine Learning?

A bias in activation function is a constant value that allows a function (f(x)) to move so that it can fit a pattern better. An example of this is a straight-line y = ax + b where b is the bias. If there is no bias, the straight line will pass through (0,0) and may not be the line of best fit for a group of continuous data. The bias, b, allows the line y = ax + b to move up or down or y = a(x+b) to move right or left so that it can be a line of best fit for any data patterns.

Optimization of the Neural Network

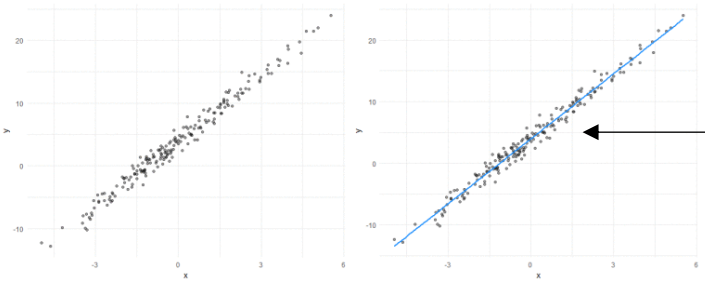
My project: Handwriting recognition research project

Every Neural network has an optimization algorithm at its core. This is because no machine learning algorithm is a 100% accurate and so there should be some measure in place so that it improves and can be more accurate than the last time it was tested.

Since there are many neural networks, they have many different optimization algorithms. A simple and common example of an optimization is using gradient descent (which involves calculus) also known as minimizing the cost function (or loss function).

Minimizing the Cost/loss Function

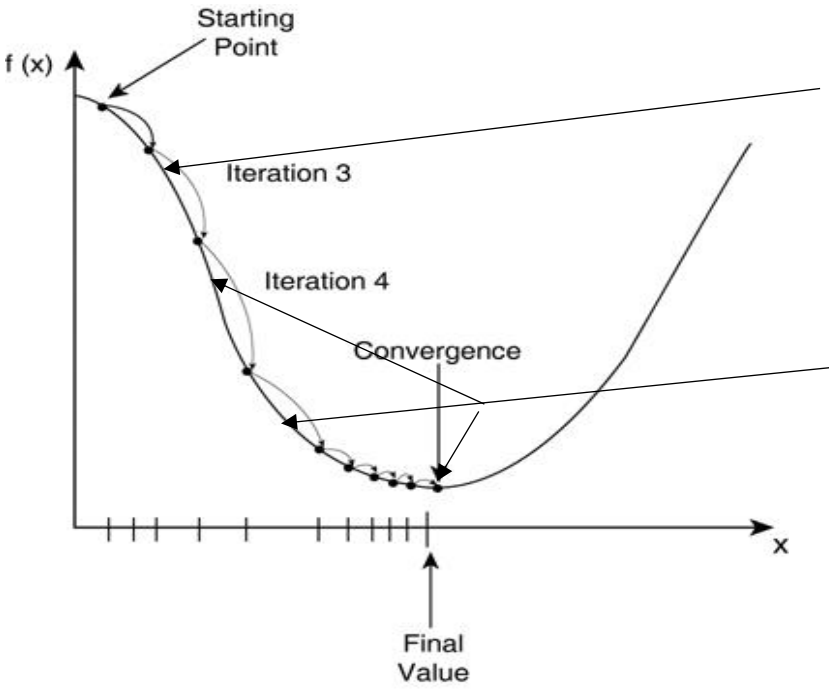
In a traditional neural network, the cost function is a measure of how bad/inaccurate a machine learning model is at estimating relationships between input and the correct result (Machine learning fundamentals (I): Cost functions and gradient descent | by Conor Mc. | Towards Data Science). This function can be used to improve the accuracy of a model. The cost function is usually calculated by finding the difference in the predicted result and actual result but there can be other complex mathematical expressions involving prediction and actual result. This function can be plotted as a graph such as linear regression (line of best fit). An example of the cost function is:



This graph shows a linear regression between data plots involving 2 variables.

Since the cost function is the relationship between the actual result and the predicted result, you need to keep the cost function as low as possible. If the predicted result of the model is different to the actual result, the cost function would be high. This is where we would have to optimise the model to reduce the cost function (i.e. the difference between predicted and actual result is as low as possible). For a graph in terms of y and x, you can work out the gradient of the graph at any given point using differentiation (calculus) $f'(x) = \frac{dy}{dx}$. This gradient can be used to work out how far you would need to go to find a local minimum (i.e. lowest cost/loss for a given predicted and actual result). This can be used to teach the model how to reduce cost/loss. An example of this is:

My project: Handwriting recognition research project



At this point, using $\frac{dy}{dx}$ you know the gradient is negative.

You would keep iterating until you reach a point where $\frac{dy}{dx} = 0$ (i.e. local minimum).

What is Backpropagation?

From the explanation above, we have learnt that "training" a feedforward neural network (i.e., a network with no loops) is just calculating and minimizing the gradient of the cost function (at least with traditional feedforward neural networks). Backpropagation is an algorithm that is used to calculate the gradient of the cost function and helps us to calculate how much we need to change our activation and weights of the neural network in order to improve it for the next iteration (epoch). The algorithm moves backwards to adjust the model's weights and biases after every feedforward pass (Neural Network Layers. Understanding How Neural Network Layers… | by Farhad Malik | FinTechExplained | Medium). Backpropagation involves every node and every layer to calculate the gradient (using chain rule).

$$\frac{\partial C}{\partial x} = [\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_m}]$$

The above example is of differentiating C (the cost function) with respect to the change in input value (x). This differentiation is an example of backpropagation because it finds the sensitivity of the change in C, based on all the relations of C with all the values of x (moving backwards in the neural network).

Once you can calculate gradients using the chain rule, you can use it in various ways to improve the network. One example is below:

$$while\ (termination\ condition\ not\ met)$$

$$w := w - \epsilon \frac{\partial C}{\partial w}$$

$$b := b - \epsilon \frac{\partial C}{\partial b}$$

$$end$$

In this example, the new weights and biases are directly influenced by the differentiation of C with respect to w and b (which would involve backpropation).

(Note: Epsilon (e) means the learning rate of the neural network and it determines the network's influence).

Section 1.4.2) In-depth analysis and break down of existing neural network solutions

For Handwriting recognition with machine learning, you have to train the model on an existing dataset. For handwriting recognition, programmers use the MNIST (Modified National Institute of Standards and Technology) Classification dataset, which is a dataset of 70,000 28x28 pixel grayscale images (in the form of arrays) of handwritten numbers between 0 and 9. This is a popular dataset to test and train datasets on before moving to bigger datasets such as EMNIST(Extended MNIST which contain all the letters in the alphabet and have a much larger data sample, also it is layed out in the same format so it is interchangeable with MNIST) because there are only 10 possible outputs with this dataset (0-9 numbers) and is relatively easy to load (because 60,000 sample size is a relatively small sample size).

We can use a CNN to train a model to recognise handwritten letters, because we can use feedforward alogrithms and activation using our input (784 nodes (28*28 = 784)) with each individual pixel values of MNIST (or EMNIST dataset images) to give a feasible output (10 nodes if MNIST dataset used).

As part of my research, I will research existing solutions for CNN for handwriting recognition (using MNIST for now) so I can learn and apply this in my final solution.

Using Tensorflow (for Python):

Tensorflow is an open-source Python library that is used for machine learning and artificial intelligence. It contains many of the popular datasets pre-loaded (including MNIST) and also methods that can substantially simplify making neural networks for programmers.

My project: Handwriting recognition research project

For my research, I will analyse an existing CNN from [TensorFlow: MNIST CNN Tutorial | Kaggle](#).

```
import tensorflow as tf
import seaborn as sns
import numpy as np
import pandas as pd
```

These are the libraries that will be necessary for making a CNN in Tensorflow.

MNIST dataset pre-processing

For the neural network model to train, we need to organise the MNIST dataset in a particular order and normalize it.

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

The MNIST dataset can be downloaded from Tensorflow.

x_train and x_test contain the array of training and test dataset images (respectively) whereas y_train and y_test contain the labels (i.e., the number that a 28x28 image contains) of those respective training and test dataset images.

Tensorflow already partitions the dataset of 70,000 into training data(60,000) and test data (10,000).

Importance of splitting dataset into training and test data:

During training a neural network model, there is always a risk of underfitting and overfitting. Underfitting is when a model isn't able to capture the relationship between the training data and output at all. This usually means the neural network is too simple to capture any patterns in the dataset. You can spot underfitting when a model has low accuracy. On the other hand, overfitting is when a model represents and fits the dataset too accurately. In this case, the model will not perform well at all with unseen data. This is a sign that the model is too complicated.

To avoid overfitting and underfitting, it is important to split the dataset into training data and test data (unseen data) so not all dataset is used for training (therefore not overtraining the model) and some data (test) can be used later to test the model.

```
input_shape = (28, 28, 1)

x_train=x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
x_train=x_train / 255.0
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], x_test.shape[2], 1)
x_test=x_test/255.0
```

Each value for every index in the arrays x_train and x_test (now of shape (60000, 28, 28, 1)) is divided by 255. This converts the array from uint8 to to float64. This is called normalisation (see below).

Before (see below), x_train and x_test were a uint8 array, with shape (60000, 28, 28) and (10000, 28, 28) respectively, but after using the reshape method, x_train and x_test have the shape (60000, 28, 28, 1) and (10000, 28, 28, 1).

My project: Handwriting recognition research project

Before resizing and normalisation (for reference above)

Name	Type	Size	Value
x_test	Array of uint8	(10000, 28, 28)	[[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
x_train	Array of uint8	(60000, 28, 28)	[[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
y_test	Array of uint8	(10000,)	[7 2 1 ... 4 5 6]
y_train	Array of uint8	(60000,)	[5 0 4 ... 5 6 8]

After resizing and normalisation

x_test	Array of float64	(10000, 28, 28, 1)	[[[[0.] [0.]
x_train	Array of float64	(60000, 28, 28, 1)	[[[[0.] [0.]
y_test	Array of uint8	(10000,)	[7 2 1 ... 4 5 6]
y_train	Array of uint8	(60000,)	[5 0 4 ... 5 6 8]

What is Normalisation (on MNIST):

By default, the MNIST stores pixel values in a range of 0-255 (0 being white pixel and 1 being a completely black pixel), which is a categorical data (because it takes a finite number of possible values). Machine learning train better on continuous data so if we divide each pixel value by 255 (as done in the example above) we can get a continuous floating-point value between 0 and 1. Another popular method of normalising MNIST is to do normal distribution with a mean of zero and a unit standard deviation of 1.

```
y_train = tf.one_hot(y_train.astype(np.int32), depth=10)
y_test = tf.one_hot(y_test.astype(np.int32), depth=10)
```

Like the x_train and y_train, y_train and _y_test is currently categoric, and not continuous. To convert it to continuous, we use the one_hot() method that takes categoric values like 3 and converts it to an array of length 10 (using the depth parameter which is 10 in this example) to [0,0,0,1,0,0,0,0,0,0].

The Convolutional Neural Network

```
batch_size = 64
num_classes = 10
epochs = 5
```

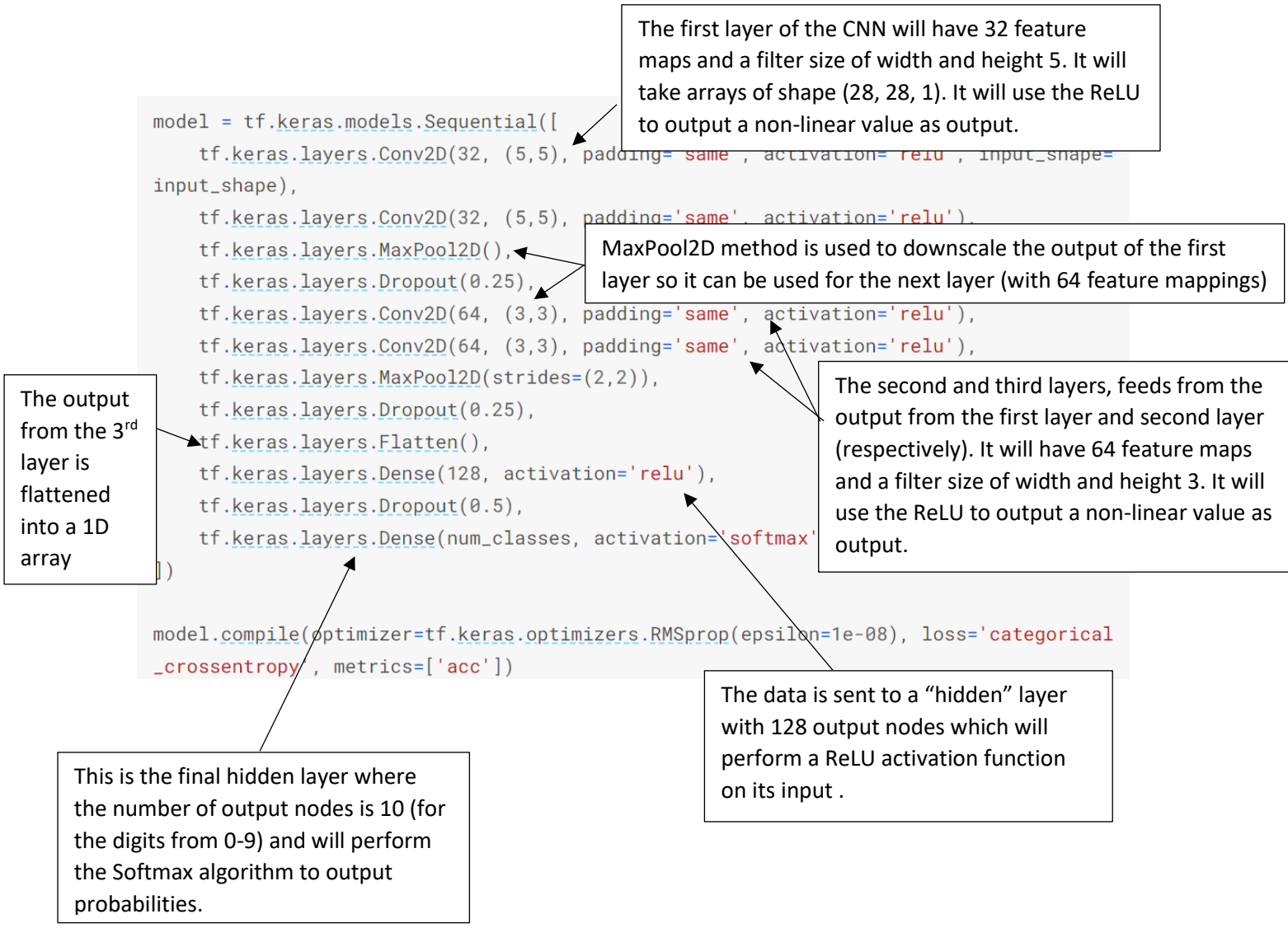
The batch size is the number of data samples that can be passed in the model in one epoch.

Num_class is the number of possible outputs there are. Since we are specifically using the MNIST dataset (number dataset) there are only 10 possible outputs (0, 1, 2, 3, 4, 5, 6 ,7 ,8 ,9). This will stay constant during experimentation.

An epoch is when the model cycles through the training data. The higher the epoch, the more cycles are done between the training dataset.

You can experiment with the epoch and batch_size to improve the accuracy of the CNN.

My project: Handwriting recognition research project



Key terminology:

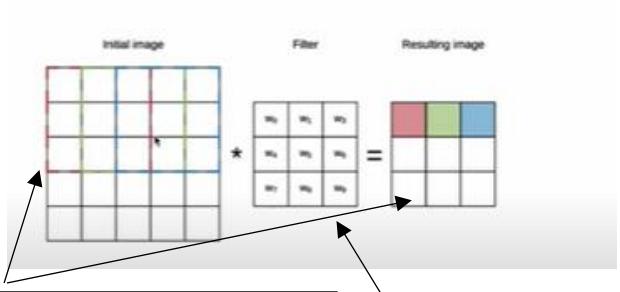
1)tf.keras.layers.Conv2D() creates a layer of convolution (i.e. takes in input parameters and produces an output layer based on the parameters).

The key (and most commonly used) parameters for tf.keras.layers.Conv2D() are:

1. filters: the number of output filters should be in the convolution (also known as feature maps)
2. kernel_size: a tuple showing the height and width of the convolution (which is 2D) window

3. padding: the above source code has padding = “same” meaning output will have the same size as input
4. activation: Non-linear functions such as ReLU, sigmoid, softmax, tanh

How a convolution layer works:



The filter does its calculation on one set of pixels and outputs a new value in the resulting image. Then it moves along to the next set of data.

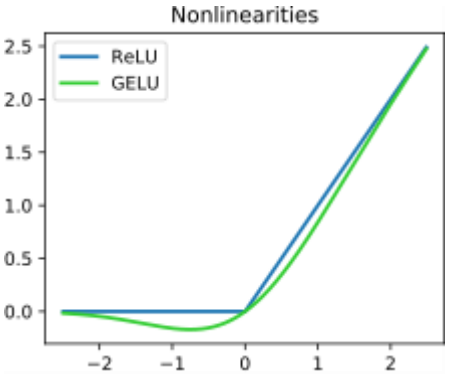
The filter is used to calculations on each pixel of the input image (such as activation (e.g. ReLU)).

2) tf.keras.layers.MaxPool2D() is responsible for downscaling the output of the convolution layer without losing the key features of the output.

3) tf.keras.layers.Flatten() converts the data from feature extraction into a 1D vector.

4) tf.keras.layers.Dense() are the “hidden layers” where calculations such as dot product between its input and kernel, happen (matix-vector multiplication).

5) The ReLU (Rectified Linear Unit) is a non-linear activation function with the equation f(x) = max(0, x) where x is the input of the neuron. It has the graph:



6) Softmax is activation function that converts a vector of numbers into a vector of probabilities. In machine learning, it can be used to normalise outputs of neurons by converting them from weighted sum values into probabilities that sum to 1. It has the equation:

My project: Handwriting recognition research project

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_split=0.1,
                    callbacks=[callbacks])
```

Finally, we need to substitute our training data into our neural network. This is why we use model.fit(). x_train is the array of training dataset, y_train is the label of the corresponding training data.

Callback is the output function (explained below).

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('acc')>0.995):
      print("\nReached 99.5% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()
```

The model outputs an accuracy of the trained model at the end of each epoch. This class instructs the model to train until the model is trained until certain threshold (in this case 99.5%).

This class is used in model.fit().

The output:

My project: Handwriting recognition research project

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/5
54000/54000 [==============================] - 10s 181us/sample - loss: 0.2200 - a
cc: 0.9317 - val_loss: 0.0437 - val_acc: 0.9872
Epoch 2/5
54000/54000 [==============================] - 5s 98us/sample - loss: 0.0748 - ac
c: 0.9782 - val_loss: 0.0313 - val_acc: 0.9915
Epoch 3/5
54000/54000 [==============================] - 6s 103us/sample - loss: 0.0600 - ac
c: 0.9828 - val_loss: 0.0294 - val_acc: 0.9917
Epoch 4/5
54000/54000 [==============================] - 5s 99us/sample - loss: 0.0522 - ac
c: 0.9854 - val_loss: 0.0321 - val_acc: 0.9922
Epoch 5/5
54000/54000 [==============================] - 6s 105us/sample - loss: 0.0507 - ac
c: 0.9862 - val_loss: 0.0454 - val_acc: 0.9903
```

The model is trained until it does 5 epoch cycles (because our input epoch = 5). The accuracy at the end of the 5th epoch is 99.03%.

My takeaways from this coded solution:

Overall, Tensorflow is great for machine learning and making neural networks quickly and accurately because it already provides the methods and frameworks for the programmer to quickly use while allowing customization and experimentation.

However, my only problem with Tensorflow, is that it makes neural network programming too easy and straightforward for me because it will not let me make my own algorithms (such as backpropagation) as it will already do that for me. I want to explore machine learning myself and make algorithms from scratch so that my research and development of this project can be meaningful to do.

Neural Network made from scratch (Python):

Since using Tensorflow is not feaseable for my project, I will analyse a neural network that was made from scratch. In this book written by Michael Nielsen Neural Neural networks and deep learning,he does a breakdown of how machine learning works by using MNIST handwriting recognition as his example. He makes a neural network from scratch (I used “network.py” from the src folder in GitHub - mnielsen/neural-networks-and-deep-learning: Code samples for my book "Neural Networks and Deep Learning"). I have not analysed each and every method in the program, only the ones that I think are important assets for my project.

The only 3 libraries that are used in this program are random, time, and NumPy.

My project: Handwriting recognition research project

```
import random
import time

# Third-party librarie
import numpy as np
```

OOP (Object Oriented Programming) is used in this program to make a CNN.

```
class Network(object):

    def __init__(self, sizes):
        """The list ``sizes`` contains the numbe
        respective layers of the network.  For e
        was [2, 3, 1] then it would be a three-l
        first layer containing 2 neurons, the se
        and the third layer 1 neuron.  The biase
        network are initialized randomly, using
        distribution with mean 0, and variance 1
        layer is assumed to be an input layer, a
        won't set any biases for those neurons,
        ever used in computing the outputs from later layers."""
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1], sizes[1:])]
```

This is the constructer method for the class Network. It takes the variable “sizes” as a parameter which is an array containing number of neurons in each layer of the Convolutional Neural Network. For example, if sizes = [10,10,30], there will be 3 layers with the input layer having 10 neurons, layer 2 having 10 nodes and the output layer having 30 neurons.

The attribute of the class “num_layers” is initialised to length of the parameter “sizes”. It stores the number of layers there are in the neural network.

The attribute “sizes” is initialised to initial parameter “sizes”, which contains individual neuron sizes of each layer in the neural network.

The attribute “biases” is initialised to an array that contains all the biases of the nodes in the model (which are created randomly using Gaussian distribution).

The attribute “weights” is initialised to an array that contains all the weights that connect each node from each layer to its adjacent layer. The weight values are created randomly using Gaussian distribution.

My project: Handwriting recognition research project

SGD (Stochastic Gradient Descent) is the main method where the neural network is trained.

```
SGD(self, training_data, epochs, mini_batch_size, eta,
    test_data=None):
"""Train the neural network using mini-batch stochastic
gradient descent.  The ``training_data`` is a list of t
``(x, y)`` representing the training inputs and the des
outputs.  The other non-optional parameters are
self-explanatory.  If ``test_data`` is provided then th
network will be evaluated against the test data after e
epoch, and partial progress printed out.  This is usefu
tracking progress, but slows things down substantially.
if test_data: n_test = len(test_data)
n = len(training_data)
for j in range(epochs):
    time1 = time.time()
    random.shuffle(training_data)
    mini_batches = [
        training_data[k:k+mini_batch_size]
        for k in range(0, n, mini_batch_size)]
    for mini_batch in mini_batches:
        self.update_mini_batch(mini_batch, eta)
    time2 = time.time()
    if test_data:
        print("Epoch {0}: {1} / {2}, took {3:.2f} seconds".format(
            j, self.evaluate(test_data), n_test, time2-time1))
    else:
        print("Epoch {0} complete in {1:.2f} seconds".format(j, time2-time1))
```

The parameters for SGD are training_data (the dataset which the neural network will train on), epoch (the number of cycles that the network will train with training_data for), mini_batch_size (the amount of training_data data that can be trained on in one epoch), eta (learning rate of the model).

The method iterates from 0 to epoch.

The training_data is shuffled randomly to eliminate any biases and introduce randomness.

mini_batches is initialised to an array where each index is of length = mini_batch_size and training_data is equally partitioned out into each index.

A for loop is created that iterates through mini_batches. For each index, the update_mini_batch method is called, with parameter as the index of mini_batches and (also eta (the learning rate)).

If test_data (the test data of the dataset) is present, then the method evaluate() is called with test_data as a parameter.

The epoch number is output at the end of each iteration

```
batch(self,
e network's
cent using b
atch`` is a list of tuples   (x, y)  , and ``e
ing rate."""
p.zeros(b.shape) for b in self.biases]
p.zeros(w.shape) for w in self.weights]
for x, y in mini_batch:
    delta_nabla_b, delta_nabla_w = self.backprop(x, y)
    nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_
    nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_
self.weights = [w-(eta/len(mini_batch))*nw
                for w, nw in zip(self.weights, nabla_w)]
self.biases = [b-(eta/len(mini_batch))*nb
               for b, nb in zip(self.biases, nabla_b)]
```

Nabla_b and nabla_w are empty NumPy arrays are created for bias and weights (using the shape of existing biases and weights)

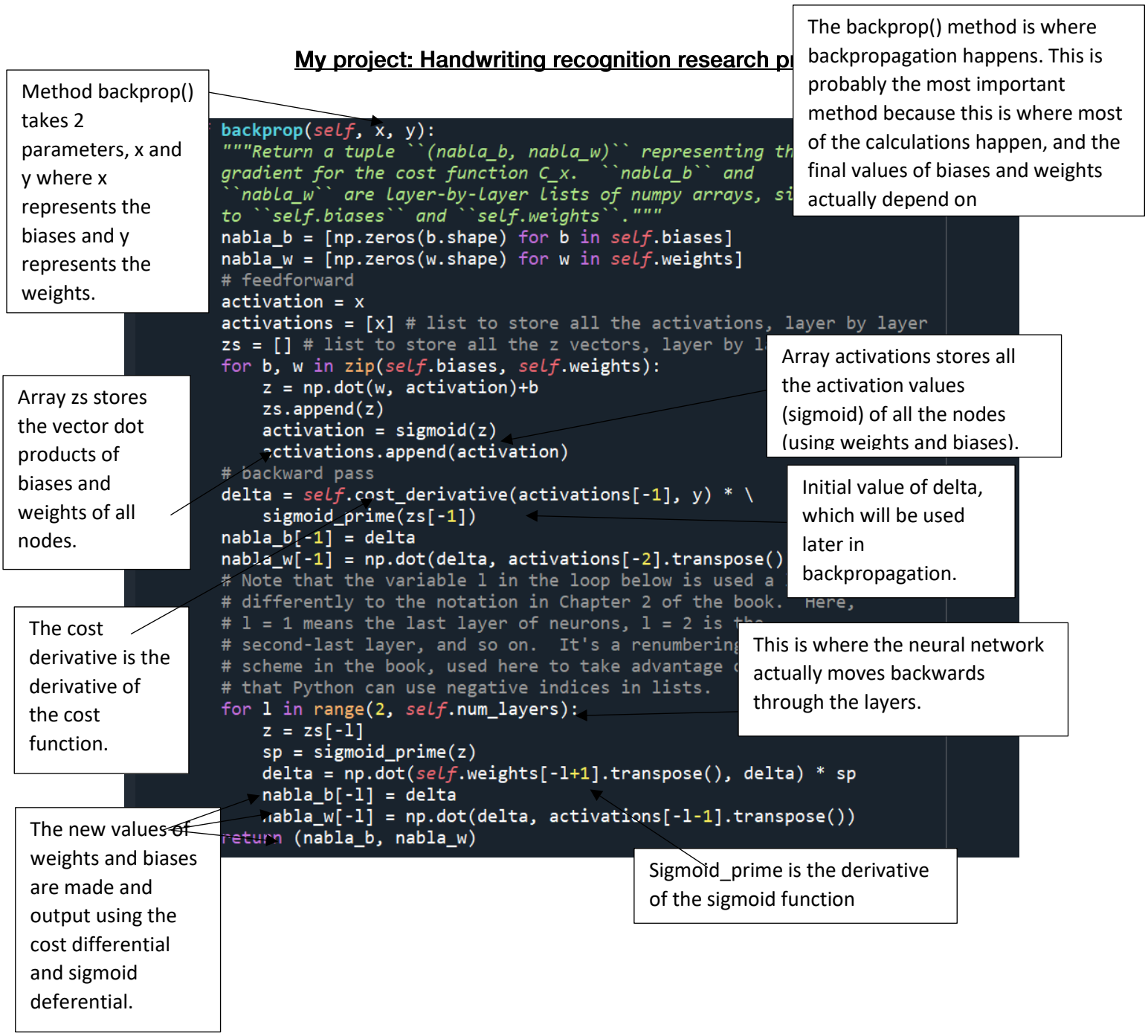
A for loop iterates within mini_batch with x meaning biases, and y meaning weights. delta_nabla_b and delta_nabla_w are assigned to the output of the backprop() method (backpropagation with x and y)

Since this is a for loop, nabla_b and nabla_w are constantly updated with and values of delta_nabla_b and delta_nabla_w (after backpropagation) are constantly appended.

Using nabla_b and nabla_w (which was constantly updated indirectly using backpropagation), the attributes weights and biases are officially updated.

The method update_mini_batch() is used to output new biases and weights (using backpropagation)and replace them with existing ones in the model.

My project: Handwriting recognition research pr



For context (Backpropagation):

In this code (made by Michael Nielsen), you have to refer to his book (referenced above) in chapter 2, he talks about Backpropagation. He explains his method for doing Backpropagation is:

1) Input training data
2) For each training data (x) index:
    a. Feedforward (i.e., traverse to the next layer) and find the formula $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^(x,l) = \sigma(z^{x,l})$ where l meaning layers 2 onwards (we don't count the input layer), w = weight, a = activation, b = bias
    b. Backpropagate the error
3) Based on the error margin, update the weights and biases accordingly.

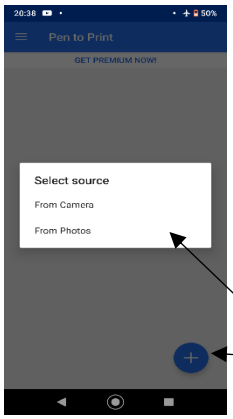
[Note to self: Write a conclusion, justifying what parts will go in your project]

Section 1.4.3) Existing product research (with graphical user interface)

To research my problem and to find out how various solutions have been implemented by programmers, I looked at some existing products that already serve a variety of stakeholders. I will use the results of my product research to model my proposed solution for this project.

First of all, during my research I found that software with handwriting recognition abilities are not as popular or widespread than I initially expected. I often found that many exemplary pieces of software (made by independent developers) were either forcing clients to pay a subscription service or make an upfront payment, which hindered my reviewing and researching process substantially because it would be very expensive for a student such as me to make payments just for ordinary research purposes.

Application 1: Pen to Print (Google Play store link: Pen to Print - Handwriting OCR – Apps on Google Play)

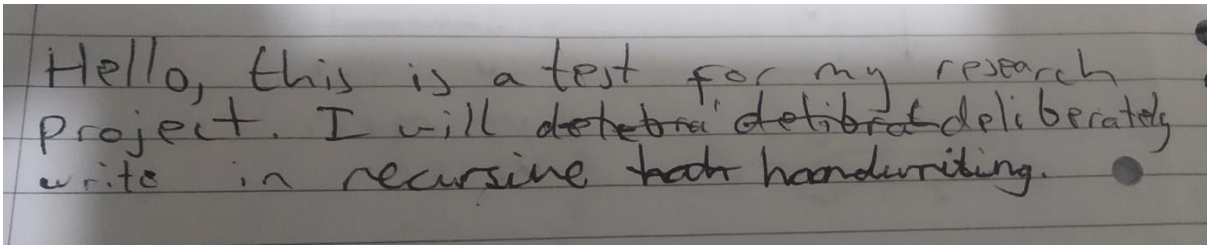


This is the first free application I found, when I researched for handwriting recognition apps. “Pen to Print” is a free app that can be found on the Google Play Store. It has in excess of 500,000 downloads.

In this application, the user has to input an image of the text that is required to be recognised and displayed as digital plain text.

The GUI of this app is very basic and far less intuitive than I would like. It doesn’t have any text or instructions to explain how to enter input, or any directions for using the app. Instead, it has one button widget at the bottom of the screen, which is much less aesthetical or intuitive (as the user is not asked or informed how to use this app).

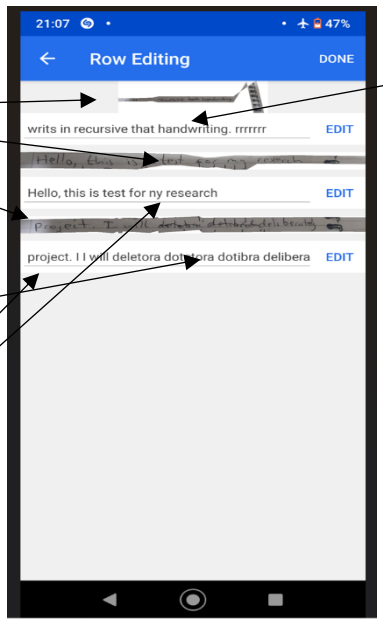
As my first input image, I inputted this image:



The problem with this image is that there is significant noise (i.e., uneven brightness across the texts) which will likely make processing much more harder. Also, I included two different types of writing styles (cursive and block script), so that I could assess how the program decomposes and interprets common styles of writing people use.

My project: Handwriting recognition research project

The output of the program was:



First of all, application managed to crop and output individual lines of code, which is very impressing considering the noise in the initial image must have hindered any line detection and therefore letter recognition.

The program could not successfully interpret my input word "deliberately", instead it output 3 nonsensical words "deletora dotibra delibera"

This program also gives the user to manually edit the output as well, so any spelling mistakes that were made during recognition (such as outputting "writs" instead of "write").

As you can see from the initial image, the words "recursive" and "handwriting" were written as recursive and I initially did not expect the program to recognise and output these words accurately. However, I was pleasantly surprised when it did recognise it and output the correct spellings. However, it has added extra words as output such as "rrrrrrrr" which were not part of initial input. Also, as a critique, I also noticed that the word "write" is output as "writs" which was not my initial input.

Parts I can apply to my solution (with justification):

One of things I really like in this project is that they have cropped and output each row of the input image alongside text boxes containing the output (which can be edited by the user). I might apply that as part of my solution because allows the user to view each line of their input and apply the necessary changes to the output which is a great design.

Overall, I am very impressed and inspired by this program because it surpassed my expectations in many areas. One of the things that surprised me was how the program could recognise cursive handwriting, which is usually hard even for humans to understand. The problem with cursive handwriting is you cannot easily decompose words into individual letters as every letter is joint up so it is hard to unpack or collect each letter. Another thing that pleasantly surprised me was the ability of the program to split the input text into individual rows even though the image had a significant noise (which usually causes hindrance for detection algorithms).

As a critique of the program, I noticed several spelling and grammatical mistakes in the output such as outputting "writs" instead of "write" or outputting "deletora dotibra delibera" instead of "deliberately". However, I did anticipate this because machine learning and especially image detection cannot be 100% accurate, especially after considering that this program was made by a small independent developer who may not have the same financial pedigree as a large software company.

The GUI of this application had significantly disappointed me, because it showcased a very basic interface that lacked text or image instructions to using this app which can potentially confuse users or dissatisfy them as the interface isn't intuitive enough.

Also, this program is advertised as "A perfect solution for students" which I would dispute because this program clearly cannot recognise each and every word and will require lots of manual input from the user if the program is used. This might be inconvenient and therefore not "perfect" for a user as advertised.

Application 2: Google Lens

My project: Handwriting recognition research project

Google Lens is a first-party app by Google that is already installed and integrated into many Android phones. The greatness about Google Lens is that it is an image recognition app, meaning a user can not only extract handwritten text, but also do translation (such as translating foreign language signs into English) and also reverse image searches (which are useful to recognise objects, people, places, etc). In this section, I will specifically analyse the text recognition part of Google lens as this is the part that I will review and potentially integrate for my project.

First of all, I was introduced to the application with an aesthetically pleasing GUI with intuitive controls:

Unlike Application 1, this app gives the user visual instructions such as “Tap the shutter button” which guides the user to access functionality of the application.

This app gives the user the choice to enter input in multiple ways such as live camera feed, but also in the form of saved images, which is very convenient as the user is not forced to act or access data in any certain way but has the freedom to choose.

The app allows, gesture control such as swiping left/right to access different functionalities such as text recognition and translation of text.

For text recognition, I decided to use the same input image as Application 1, because I wanted to test how Google lens would approach the difficulties with the input (such as noise and inconsistencies with writing styles).

The output:

Google lens displays bounding boxes around text regions in the input image.

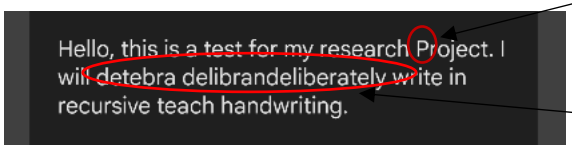
The program successfully identified the text region in the image (like Application 1) and allows the user to copy parts/all of the text which I really like because it allows the user to convert and copy any text they desire for their purpose. It allows greater control over the output than Application 1.

I was disappointed because the program did not show the output of the full text that was written and instead expected the user to copy the text.

This feature really surprised me because, Google Lens allows the user to get dictation of the input image, alongside allowing the user to copy the text to a computer.

My project: Handwriting recognition research project

The final text (I had to copy this to an empty text editor because Google Lens would not output the full text in my input):



The program capitalised the "P" in "project" which was not part of my initial input.

Interestingly, both application 1 and application 2 did not interpret the word "deliberately" correctly. Instead, it output a nonsensical phrase "detebra delibrandeliberately".

Overall, considering that the input image was full of noise and inconsistencies, Google Lens output a more accurate representation of the initial input than Application 1. Despite one major spelling mistake, the program output a grammatically accurate output.

Parts I can apply to my solution (with justification):

There are a lot of GUI features that fascinated me in this app. One feature that I am most impressed by is showing bounding boxes around input image containing text, which is a visual cue for the user what sections of the input is recognised and what hasn't. I also liked how the app allowed the user to copy any sections of recognised code and paste it in other applications. I don't think I can implicate that in my program because I don't think I can make a text-copying prototype in the timescale I have to finish this project.

Overall, Google Lens has a much better implementation of handwriting recognition algorithms and app functionalities (alongside user interface) than "Pen to Print". First of all, Google lens allows live camera feed as input meaning the user can hover their smartphone over their handwritten text and expect instant output as digital text, whereas "Pen to Print" only accepts standalone images as input which is much more inconvenient in terms of user experience (as they need to save an image each time they need their input). Also, Google Lens has a much more intuitive and inclusive user interface than "Pen to Print" as it gives the user more choices with features such as text recognition, translation and interface features such as gesture controls, whereas Pen to Print, only has text recognition and lacks in UX design.

Conclusion

The purpose of my product research was to mainly assess how modern handwriting recognition algorithms work and how realistically they can recognise and output digital text based on input data. After seeing how products made by independent developers and big corporations both failed to carry out text recognition with 100%, I will reduce my expectations for my proposed solution, as I know my implementation will not be any major overhaul over existing solutions but just an attempt at making a recognition software that can try and serve its stakeholders.

Also, I did not have any major takeaways for the GUI implementations for the products I researched because they are both mobile apps and I don't think that I can model this for my desktop solution. However, I can implement many features that were used in existing products such as bounding boxes around text regions in images which can be used as visual feedback for the user.

Section 1.5) Initial Proposed Solution (based on research)

Given the timescale for this project (3 months), I will use Python and specifically the Tkinter library for my graphical user interface and machine learning algorithm because Python is a hugely popular language when it comes to machine learning and has a wide range of support for handwriting recognition online (on popular websites such as Stackoverflow and Github) which will be beneficial for me as I do not have much experience with machine learning and I will likely stumble in my programming so having support online is very handy.

Tkinter is an excellent library for GUI and UX design due to its ease of use in terms of programming and you can program a lot of complex designs fairly quickly and easily because it is straightforward to learn and is well documented so if there are any bugs in my program, I will have a lot of support online to help me fix it. However, one of the fundamental problems with Tkinter is the lack of customization with widgets because you cannot change the look or feel of the widgets (such as buttons) and it might make the program look and feel old fashioned or out of date (aesthetically).

Using Python will mean my program cannot be portable, for example, it will require the users to install Python on their computer devices with PIP to install external libraries such as numpy (which is not included in the Python standard library) which might be too confusing or impractical for my users. Also, this means that my program cannot be used on mobile devices, which makes my solution ergonomically unfeasible. As seen in Product Analysis, one of the features both Android apps had was using the device camera to take a photo and use it as an input. However, I cannot implement this feature in my desktop project because most computers do not either have a camera or have a low-resolution webcam that will be impractical to take pictures and use as input (because it will likely produce a lot of noise and will require heavy post image production) because it will produce inconsistent results for recognition.

With my initial research into machine learning and existing applications, I want my final solution to use a CNN (Convolutional Neural Network) because it is a fairly efficient neural network as compared to other neural networks (such as RNN) and is the most-suited to my problem because, the CNN can take pixel values as an input and output feasible activation for each node in hidden layers which can then be used to give an accurate output (i.e. letter).

Other key features I want in my proposed solution are:

1. As input image for the program, only accept ".JPEG" or ".PNG" images because it is much more straightforward to work with those images when working in OpenCV, Tkinter and other libraries.
2. Display bounding boxes around recognised text in the image

My project: Handwriting recognition research project

3. Display a cropped image of each individual line/row in the input image, alongside text output for that line/row (in a text box so it can be manually edited by the user in case there are incorrectly recognised letters)
4. Make a final output text file that contains the recognised text (from input)
5. Allow manual editing of bounding boxes so any unnecessary boxes created (due to noise) can be eliminated.
6. Create scrollbars (horizontal and vertical) and zoom scale to zoom in/out large and small images.

Hardware and Software requirements

1. Hardware:

   A computer that is capable of running Python 3 (or above) and has at least 4GB RAM and at least excess of 4GB storage (this is because the handwritten letter dataset is a very large file (3GBs) and requires a lot of RAM to the load for the neural network).

2. Software:
   - Python 3x
     - [External library to install using PIP] NumPy
     - [External library to install using PIP] OpenCV (CV2)
     - [External library to install using PIP] pickle
     - [External library to install using PIP] gzip

Key features of my solution

Functionality: Image pre-processing

Feature	Explanation	Order of priority (High/Medium/Low)
Remove noise from input image	Using the OpenCV (CV2) module, remove unnecessary noise from the image so it doesn't hinder text detection.	Medium

Functionality: Image Recognition

Feature	Explanation	Order of priority (High/Medium/Low)
Detect individual letters	Using CV2 contour detection, detect and save an image of each letter detected in input image. This can be used for neural network.	High
Make image contours thicker	If the letters are faint, they will be less likely to be recognised by the neural network, so make the contours thicker.	High
Partition detected letters into rows	Using coordinates of each detected letter, work out which row/line each letter go in and organise in a list.	High
Group detected letters to form words	Using distances between each letter, group all the detected letters into words (use a 2D array, to group rows and words)	High

My project: Handwriting recognition research project

Functionality: Neural Network

Feature	Explanation	Order of priority (High/Medium/Low)
Make an accurate neural network from scratch that can recognise letters	This is the foundation for a successful project. In order for my program to successfully recognise and output a file (based on handwritten data), I will need a neural network that is accurate and efficient in recognising digits otherwise my output for the user will be erroneous.	High
Backpropagation algorithm is efficient and accurately updates the neural network	The single part that makes a neural network accurate is backpropagation. Backpropagation is the part of a neural network that is responsible for going back through the neural network and actually updating all previous weights and biases in order for a more accurate neural network for next epoch	High
Split the image dataset into training and test dataset to avoid overfitting	As I have explained in **Section 1.4.2,** overfitting is when a model represents and fits the dataset too accurately. Therefore, the model will not perform well at all with unseen data. This will reduce the accuracy of the neural network, which in turn will give me erroneous results for output. To prevent overfitting, I must split the image dataset (EMNIST) into training and test data so I am not using all of it for training and other data can be used for testing later on.	Low

Functionality: Graphical User Interface

Feature	Explanation	Order of priority (High/Medium/Low)
Dedicated widget to show input image	This is necessary because there will be various operations that will be performed around the input image (such as showing text region using bounding boxes or zoom functions, but most importantly, recognition of all the text in the image). Therefore, it is important to make a widget that is dedicated to show the input image.	High
Accept only “.JPEG/JPG” and “.PNG” files	Due to the limited time scale of this project, I cannot predict how other file formats such as Bitmap(.BMP), GIF, etc. would perform in my image processing, and neural network methods, therefore I will only explicitly only accept .JPEG/.JPG or .PNG files because I will only test my network using these image formats.	High

Add zoom in and out operations for image	Input images can be big or small. I will require a zoom functionality that can allow the user to view the image according to their preferences.	High
Show bounding boxes around each detected letter	This is a visual output that I was inspired from Google Lens and Pen To Print (in Product Analysis) to make. This will show the user what regions of the text has been identified (and what data will feed into my machine learning algorithm).	High
Show snippet of each row of input with text output beneath it.	I was inspired by Pen To Print in the Product Analysis to add this functionality to my project. Showing snippets of each row of input alongside with its generated output (which the user can change) is useful because it allows the user to view each line of their input and apply the necessary changes to the output.	High
Make an output text file	Once my program has recognised and generated text (using the given input image), an output file will be created, which the user can later copy and paste into their other documents.	Medium

Limitations of my solution (with justification):

1) The first limitation of my project is that it will be made in Python, meaning that the user of this program will have to download Python and all the necessary libraries before they can use my program. This also means they are forced to use a computer in order to access the program and cannot use their phones (which is more portable and convenient). Therefore, my program is not portable which is a downgrade in terms of user experience.
2) Since I will be designing and implementing a CNN from scratch instead of using any existing 3rd party programs for text recognition, my program will not have a good accuracy with recognising letters. This means frequent user interaction will be required to edit and correct text files that were generated by my program. This is very inconvenient for user experience.
3) The main purpose of my program is to recognise letters. Therefore, my program is not good at eliminating noise from input images, so it might use the noise as input for text recognition which will generate erroneous and non sensical text as an output.
4) Since I will be using Tkinter in Python for making my GUI (Graphical User Interface), I will not be able to customise the shapes and looks of certain widgets such as buttons and text boxes. This reduces my scope to experiment with the user interface and make it stand out.

Section 1.6) Success Criteria for the solution (during development) which are measurable:

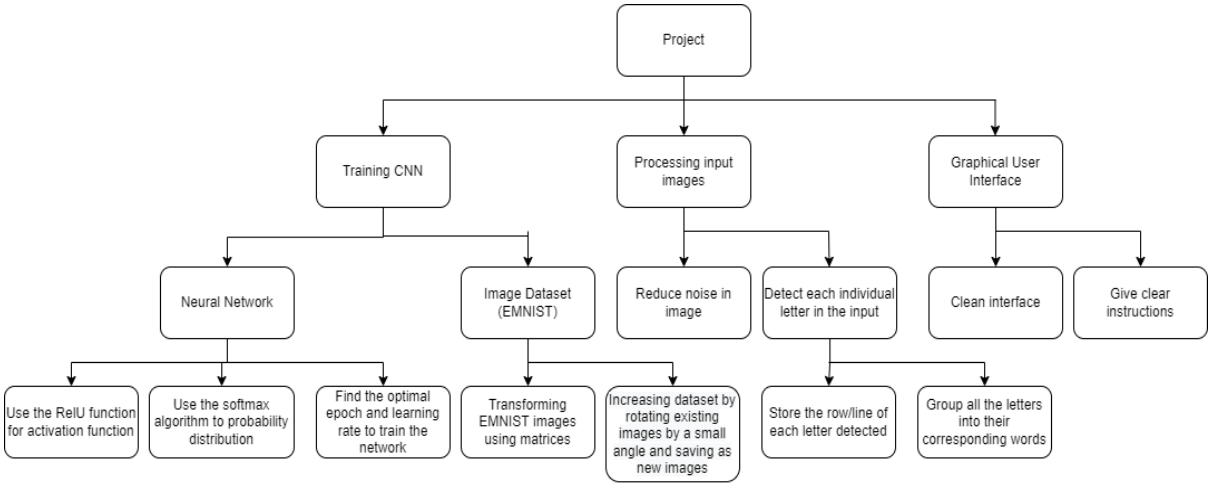
Objective	How to measure

My project: Handwriting recognition research project

Having a clean user interface by having no more than 4 buttons on one window	Screenshot the home screen where the user is given choice of what functionalities to use.
Have 2 tabbed windows	Screenshot the top of the window where tabs are available (and show that they work).
Load a dropdown menu where the user can choose to open an image	Screenshot the dropdown menu with proof of it working.
Have a widget that covers 50% or more of the window that shows the input user image	Screenshot the image widget with proof of it working with any .JPEG/.JPG and .PNG file.
Have a zoom functionality to enlarge input image to 200% or shrink to 0 % (inside widget)	Screenshot the zoom slider working.
Have scrollbars (horizontal and vertical) to navigate user input image in widget	Screenshot the scrollbars working in relation to the image widget.
Have a 1 button to detect letters in the user input image	Screenshot proof of the detection algorithm working on several images.
Have 1 button to output digital text generated by the CNN (redirects to a new tab)	Screenshot the new tab that displays the output.
In the output, display snippets of each row of input text (if there were paragraphs in input text) alongside generated text output just below. It should cover at least 40% of the screen	Screenshot proof this functionality with several input images.
Have scrollbars (horizontal and vertical) to navigate rows in output window	Screenshot the scrollbars working in relation to output text of each row.
Generate a .txt output file on a button click	Copy/Paste output text from the output file and screenshot the use of a button click for producing an output file.

Section 2) Design Section:

Section 2.1) Problem decomposition for design (with justification):



My project: Handwriting recognition research project

In this section, I have decomposed my project into 3 main components:

- Training CNN (Convolutional Neural Network) – this is the foundation of my project because the final output depends on the neural network model I design and therefore it is crucial to design accurate and efficient algorithms so that the optimal outputs in recognising of my input can be delivered.
- Processing input image – For the optimal solution to be found, the input must be cleaned and decomposed into smaller parts, which can then be used in my CNN. Since neural networks are sensitive to noise in images, I will need to denoise the input image to minimise error margin. Also, since I am making my neural network from scratch, it will only recognise only one letter at a time, so I need to design an algorithm that can detect text letters and recognise one letter at a time.
- Graphical User Interface – The GUI is where the user can communicate with my recognition program, so it is crucial to make a clean, and straightforward interface where the user can operate and control the program with ease.

Justification:

I have decomposed my project into these 3 specific components because:

1) Training CNN: My letter recognition depends on my CNN development; therefore, I need to design and develop an algorithm for my CNN. Since my GUI functionalities depend on letter recognition, I cannot make a GUI or its functionalities without making a working letter recognition algorithm.
2) Processing input image: I have partitioned this part of my program into a separate decomposition section because this is the part of my program which outputs the data that will feed into my neural network for letter recognition. It is crucial that this part of my code can process and manipulate user input data and output a decent, readable input for my neural network, otherwise my whole program and project will not work at all. Also, I cannot start developing image processing algorithms until I make my CNN algorithm, as I need my CNN to test my image processing algorithms on (to work out optimal solutions for processing individual letters). Therefore, image processing is a separate section where I can construct a good algorithm for detecting letters and manipulating it so that it can be feed in for my neural network.
3) Graphical User Interface: This is the last section I will work on, as no part in my program depends on me making the GUI, and the purpose of the GUI is only to show and allow the user to access a functionality.

Extra Note:

In this project, I plan to use OOP (Object Oriented Programming) wherever possible because it is many advantages such as:

- Modularity: In OOP, all the methods are modular meaning every method in a class can be edited or manipulated without affecting any other method directly. This also means troubleshooting is easier, because a part of the program can be tested out separated.

- Organisation: If you have a lot of functions, you can rearrange them in multiple classes. This would make it easier for searching methods, especially if you have a large programming file with lots of lines, because you can just visit the appropriate class and find methods there.

Section 2.2) Algorithms of each key part of solution

Section 2.2.1) Training CNN

Probably the single most important part of this project is training a neural network and using it to recognise handwritten inputs.

Parts of design in my Neural Network (training):

In my Neural Network training class, there are 3 main methods:

1. Gradient Descent (GD): This is the main method where training of the NN starts. Initially, the feedforward method is used to set initial values to all weights and biases in the network. Gradient Descent method then iterates (epoch), in which the backpropagation method is used, with the aim of making the neural network more accurate with each iteration. The number of iteration (epochs) is a user specified value. It outputs an accuracy (the cost which is expected outcome – the neural network outcome) as a percentage. I can use this to further amend or change the neural network (to make it more accurate).
2. Backpropagation: The algorithm takes the cost (or loss) function (which is the difference between the expected outcome and neural network outcome) alongside partial differentials of all the weights and biases in the neural network (chain rule), to output a values or "nudges" that can be made to the initial weights and biases to give a more accurate neural network.
3. Updating parameters: Using the outputs from backpropagations, the appropriate changes are made to the initial weights and biases of nodes. With each iteration (epoch) of gradient descend, this will help reduce the cost of the neural network.
4. Feedforward: As the name suggests, it "feeds" forward the neural network, meaning based on the input array, the neural network will update its weights and biases. This will then be used in activation of the neural network (such as sigmoid or Softmax) from which the result can be deduced.

Detailed Decomposition of my neural network:

[Note: For neural network part I have not put detailed notes, instead I have used a lot of detailed comments]

Also note: for backpropagation() I have used a solution from the website Simple MNIST NN from scratch (numpy, no TF/Keras) | Kaggle because the neural network in the website closely matched mine (because it has a dedicated layer for softmax and so does mine) and so I implemented their backpropagation algorithm. It is very difficult to design and make a backpropagation algorithm from scratch as it requires a deep understanding of calculus and neural networks. Due to the time scale of this project, I couldn't develop the backpropagation algorithm myself.

Pseudocode with detailed comments:

Class Network

My project: Handwriting recognition research project

```
        private size_layer1
        private size_layer2
        private size_layer3
        private size_activation_sigmoid
        private size_activation_softmax
        public weights_layer2
        public biases_layer2
        public weights_layer3
        public biases_layer3
        private activation_sigmoid
        private activation_softmax
        public procedure new() //constructer method; I will make 3 layers, first (input) having 784 nodes, second (hidden layer) having 10 nodes and third (output) having 10 nodes. I will initialise the nodes with random values
                size_layer1 = (784, 1) //input layer
                size_layer2 = (10, 784) //hidden layer (with sigmoid activation)
                size_layer3 = (10, 10) //output layer (with softmax activation applied to it)
                size_activation_sigmoid = (10, 1) // stores sigmoid activation of all 10 nodes in layer2
                size_activation_softmax = (10, 1) // stores sigmoid activation of all 10 nodes in layer3
                weights_layer2 = []
                for x = 0 to 10-1
                        temp1 = []
                        for y = 0 to 784-1
                                temp1.append(random.randomValue(0,1) ) //setting initial node values for layer 2
                        weights_layer2.append(temp1)
                        next y
                next x
                biases_layer2 = []
                for x = 0 to 10-1
                        biases_layer2.append(random.randomValue(0, 1)) //setting a bias for each node in layer 2
```

```
                next x
                weights_layer3 = []
                for x = 0 to 10-1
                        temp1 = []
                        for y = 0 to 10-1
                                temp1.append(random.randomValue(0, 1)) //setting initial node values for layer 3
                        next y
                        weights_layer3.append(temp1) //setting initial node values for layer 3


                next x
                biases_layer3 = []
                for x = 0 to 10-1
                        biases_layer3.append(random.randomValue(0, 1)) //setting a bias for each node in layer 3
                next x
        endprocedure
        public function dot_product(array1, array2)// this is vector dot product (I will use this in feedforward() later on)
                output_array = []
                for x = 0 to len(array1)-1
                        temp1 = []
                        for index = 0 to len(array1[x])-1
                                multiply1 = array1[x][index] * array2[x][index] // the element in array1 is multiplied by its corresponding element in array2
                                temp1.append(multiply1)
                        output_array.append(temp1)
                        next index
                next x
                return output_array // the output of dot product between array1 and array2
        endfunction
```

My project: Handwriting recognition research project

[Note: the method matrixMultiplication is added after I started my development section]

```
public function matrixMultiplication(mat1, mat2) //parameters mat1 and mat2 represent mat1 multiplied by mat2 (in that order)

        row_mat1 = len(mat1)

        row_mat2 = len(mat2)

        column_mat1 = len(mat1[0])

        column_mat2 = len(mat2[0])


        if column_mat1 != row_mat2 then //this is validation, if the number of columns of mat1 doesn't equal to the number of rows of mat2, then matrix multiplication cannot be done, so output a Boolean False.

                return False

        else

                final_dim = (row_mat1, column_mat2) // the final dimensions of matrix multiplication always has the rows of mat1 and columns of mat2

                final_list = []

                for x = 0 to row_mat1-1: //making an empty matrix of dimensions final_dim

                        temp_row = []

                        for y = 0 to column_mat2-1:

                                temp_row.append(0) // columns created with initialised value of 0

                        next y

                        final_list.append(temp_row)


                for x = 0 to row_mat1-1: //since I need multiply all the elements in a row of mat1 by the elements in every column of mat2, I will need a nested for loop with one iterating in the number of rows of mat1 and one iterating in the number of columns of mat2

                        for y = 0 to column_mat2-1:

                                total = 0 //for each mat1 row that is multiplied by mat2 column will need a way to work out the running sum (cumulative sum).

                                for z = 0 to column_mat1-1 // z keeps a track of what mat1[row mat1][column mat 1] is being multiplied with mat2[row mat2][column mat2]

                                        total = total + mat1[x][z] * mat2[z][y] // running total
```

```
                    next z
                    final_list[x][y] = total // the dimensions of final_list has already been made, so I just substitute matrix multiplication sums in its right position in the final_list
                next y
            next x
            return final_list // result of matrix multiplication is returned.
        endfunction

    public procedure feedforward(input_layer)
        dot_product_with_input = dot_product(weights_layer2, input_layer) //dot product between layer 1 and layer 2 before sigmoid is applied, the resulting array will be dimension (10, 784)
        dot_product_with_added_bias = []
        for x = 0 to len(dot_product_with_input)-1
            for y = 0 to len(dot_product_with_input[x])
            dot_product_with_added_bias.append(dot_product_with_input[x][y]) + biases_layer2[x] // bias is applied to every 784 elements of each node (total 10)
            next y
        next x
        //the sigmoid is added here
        new_weights = []
        for x = 0 to len(dot_product_with_added_bias)-1
            temp1 = []
            for y = 0 to len(dot_product_with_added_bias[x])-1
                temp1.append(sigmoid(dot_product_with_added_bias[x][y])) // the sigmoid of each element inside each node is worked out
            new_weights.append(temp1) //the sigmoid values are saved in new_weights
            next y
        next x
        activation_sigmoid = new_weights // results of sigmoid
        // the softmax is added here
```

```
matrix_multiply_layer2_activation_layer1 = matrixMultiplication(weights_layer3, activation_sigmoid) //matrix multiplication between weights of layer 3 and sigmoid values so I can combine both sets of informations
                matrix_with_added_biases = []
                for x = 0 to len(matrix_multiply_layer2_activation_layer1)-1
                        temp = []
                        for y = 0 to len(matrix_multiply_layer2_activation_layer1[x])-1
                                temp.append(matrix_multiply_layer2_activation_layer1[x][y] + biases_layer3[x]) // biases of layer 3 are added to each element of the results of matrix multiplication
                        next y
                        matrix_with_added_biases.append(temp)
                next x
                final_probabilities = softmax(matrix_with_added_biases) // softmax is applied to the matrix multiplication (and added bias)
                activation_softmax = final_probabilities // the final array is a list of probabilities that add up to 1, and contain the output of the neural network
        endprocedure
        public procedure backpropagation(inputArr, inputLabel)
                dCost_dx = []
                for x = 0 to len(activation_softmax)-1
                        dCost_dx.append(activation_softmax[x] - inputLabel[x]) // finding the derivative of cost between predicated output and actual output
                next x
                dWeights3 = matrixMultiplication(dCost_dx, activation_sigmoid) // this array will contain the values that need to be incremented to weights for layer 3 to make it more accurate
                dBias3 = dCost_dx // this array will contain the values that need to be incremented to biases for layer 3 to make the neural network more efficient
                dW3_dCost_dx = dot_product(weights_layer3, dCost_dx) // this dot product will be required in the calculation for dWeight2
                sigmoid_prime_list = []
                for x = 0 to len(weights_layer2)
                        for y = 0 to len(weights_layer2[x])
```

```
                    sigmoid_prime_values = sigmoid_prime(dot_product(weights_layer2[x][y], inputArr[x])) // sigmoid_prime_list array will contain the values of derivative of sigmoid, which will help access the weights of layer 1
                next y
                sigmoid_prime_list.append(sigmoid_prime_values)
            next x
            dActivation_dLayer2 = matrixMultiplication(dW3_dCost_dx, sigmoid_prime_list)
            dWeight2 = matrixMultiplication(dActivation_dLayer2, inputArr) // this array will contain the values that need to be incremented to weights for layer 2 to make it more accurate
            dBias2 = dActivation_dLayer2 // this array will contain the values that need to be incremented to biases for layer 2 to make the neural network more efficient
            return dWeights2, dWeights3, dBias2, dBias3 // outputs the values that need to me incremented onto weights_layer3, weights_layer2, biases_layer3, biases_layer2
        endprocedure


        public procedure gradient_descent(learning_rate, epoch, inputArrays, inputLabels)
            for x = 0 to epoch-1 // this will loop the training of the neural network until the end of epoch is reached
                prediction_count = 0 // this will be used to work out accuracy of the neural network after each epoch (x) is finished
                for y = 0 to len(inputArrays)-1 // within each epoch, the neural network is feeded with each image array from inputArrays
                    feedforward(inputArrays[y])
                    dWeights2, dWeights3, dBias2, dBias3 = backpropagation(inputArrays[y], inputLabels[y])
                    for index = 0 to len(weights_layer2)-1
                        for index1 = 0 to len(weights_layer2[x])-1
                            weights_layer2[x][y] = weights_layer2[x][y] + learning_rate*dWeights2[x][y] // weights of layer 2 are updated using backpropagation outputs
                        next index1
                    next index
                    for index = 0 to len(weights_layer3)-1
                        for index1 = 0 to len(weights_layer3[x])-1
                            weights_layer2[x][y] = weights_layer3[x][y] + learning_rate*dWeights3[x][y] // weights of layer 3 are updated using backpropagation outputs
```

```
                    next index1
                next index
                for index = 0 to len(biases_layer2)-1
                    biases_layer2[index] = biases_layer2[index] +
learning_rate*dBias2[index] // biases of layer 2 are updated using backpropagation outputs
                next index
                for index = 0 to len(biases_layer3)-1
                    biases_layer2[index] = biases_layer3[index] +
learning_rate*dBias3[index] // biases of layer 3 are updated using backpropagation outputs
                next index
                result_Label = -1 // result_Label just stores the desired output of the
neural network
                for index = 0 to len(inputLabels[y])-1
                    if inputLabels[y][index] == 1 then
                        result_Label = index
                    endif
                next index



                    if numpy.argmax(activation_softmax) == result_Label then
                //numpy.argmax just outputs the index of the highest probability index in
                activation_softmax
                        prediction_count = prediction_count + 1 // if the neural
network correctly recognises the image, increment the prediction count
                    endif
                print("Iteration: " + str(x) + " accuracy: " +
str(prediction_count/len(inputArrays))) // at the end of each epoch, the accuracy is output
            next x
        endprocedure
endclass
```

Section 2.2.2) Image processing

Image pre-processing

This section of my problem decomposition will only have one component:

1. Image denoising – Using OpenCV, I will perform existing denoising algorithms (such as Binary thresholding) to remove all unnecessary detail from the background of the input without affecting the foreground (the handwritten text). This will aid during text detection (so unnecessary data isn't detected as text to be recognised) and remove any error margins.

Algorithm of image pre-processing (in words):

- Image denoising:
  1) Load the input image (**image_input**)
  2) Convert **image_input** to grayscale because thresholding algorithms only accept grayscale images (also it eliminates further complexities caused by individual RGB values of the original image, instead working with black and white pixel values is much simpler). Call this image **image_gray**
  3) Apply Binary Thresholding (explained below) on **image_gray**

Thresholding (for reference above):

Thresholding is a technique in OpenCV, which is used to remove noise from an image. It involves comparing each pixel value in an image against a threshold value (provided by the programmer). If the pixel value is smaller, then it is set to 0, otherwise it is set to a maximum value (255 in most cases). The disadvantage of using thresholding is that it outputs a grayscale image and not an RGB image which means that it will lack colour details. However, for my project I do not need to know colour information for an input, because analysing black and white images for machine learning is much less complicated than using RGB images.

For my project, I will use Binary Thresholding, which is the most basic thresholding algorithm. If pixel value is bigger than a threshold, set the value to 255, otherwise 0.

Pseudocode:

```
image_pixel_value = [[1,3,233,33,211,114,….],[211,134,65,69,78,111,…..], ……] // this is an example image where each index represents a row in the image and contains pixel value of each pixel in the image.

Function binaryThreshold(image_pixel_value):

        threshold = 50

        for x = 0 to image_pixel_value-1

                for y = image_pixel_value[x]-1

                        if image_pixel_value[x][y] < threshold then

                                image_pixel_value[x][y] = 0

                        else

                                image_pixel_value[x][y] == 255
```

```
                endif
            next y
        next x
        return image_pixel_value
endfunction
```

Pseodocode for image preprocessing:

```
class image_preProcessing
        private image_input
        public procedure new(image_as_array): //constructor class
            image_input = image_as_array
        public function RGB_to_Grey(img1): //converting image to grayscale (for thresholding later)
            image_gray= OpenCV.cvtColorGray(img1)
            return image_gray
        endfunction
        public function binaryThresholding(threshold_value, image_array): //thresholding (In my actual program I will use the OpenCV thresholding algorithm instead of this basic version)
            for x = 0 to image_array -1
                for y = image_array [x]-1
                    if image_array [x][y] < threshold_value then
                        image_array [x][y] = 0
                    else:
                        image_array [x][y] == 255
                    endif
                next y
            next x
            return image_array
        endfunction
        public function denoise(): //this is where all above methods are invoked for a cleaner image
            image_gray = RGB_to_Grey(image_input)
            image_denoised = binaryThresholding(100, image_gray)
```

```
            return image_denoised
        endFunction
endClass
```

Image Recognition:

Image Recognition is a crucial part of my project. This is where the input image is decomposed into individual letters and remodified (e.g., the contour of faint lines are made thicker artificially so features are not lost when the letters are downscaled to 28x28 images for recognision). Also, while the input is decomposed into letters, they also need to be used to form digital text after the recognision phase (e.g., using the coordinates of each letter image to group it into words and rows (lines) when digital text is generated). In Image Recognition, I need to make algorithms for:

1) Detect individual letters – Using OpenCV contour detection, I can detect and map out the coordinates of each letter in the input for the recognition phase and post-recognision phase (generating digital text (words and lines of text) using coordinates as explained above)
2) Make letter contours thicker – If the letters have faint lines, it can be hard for my CNN model to accurately recognise the letter (which can cause problems in generating output), so using OpenCV I can artificially make the lines thicker.
3) Partition letters into lines/rows – This is required post recognision phase, because I will need to use my coordinates for each letter image (and hence, its CNN output) to determine its relative position in the output text. I will need to work out what letters are in each row of the input text (e.g., written paragraphs) using coordinate geometry.
4) Group letters to form words – Similar to 3), I will need to use coordinate geometry to determine distances between each adjacent letters and hence determine when one word starts and ends.

Algorithms of Image Recognition sections:

- Detect individual letters
    1) Use **class** image_preProcessing to denoise the input image (call it **imgInp**)
    2) Use Binary Inverse Thresholding from OpenCV (inverts all black pixels (in my input this will be the text) to white, and the white pixels (the background) into black). This highlights the text and makes it easier for contour detection. Therefore this is optional, but is strongly recommended. Save this image (where Binary Inverse Thresholding is done) as imgBinInv
    3) Use OpenCV findContours() method to find all the regions in the text that contain objects (in my case, the individual letters). Assign this array of contours as contours
    4) For each index in contours:
        1) Use contourArea() method in OpenCV, to find the area of each object detected, if it is smaller than a certain threshold then it is probably noise and so should be ignored.

2) If it is bigger than the threshold, find the x, and y coordinates of the contour and using that crop a rectangle containing the object. Save this as an .PNG image.

Pseudocode:

```
function detect_Letter(inp_image)
    img_preprocessing_object = img_preprocessing(inp_image)
    denoised_image = img_preprocessing_object.denoise()
    invBinInv = OpenCV.binary_thresh_inv(denoised_image)
    contours = OpenCV.findContours(invBinInv)
    list_of_coors = []
    for index = 0 to len(contours)-1
        if OpenCV.contourArea(contours[index]) < 50 then
            x += 1
        else:
            x,y,w,h = OpenCV.boundingRect(contours[index]) //OpenCV method that finds makes a rectangle around the contour object and outputs the (x,y) coordinates of the rectange and the width and height.
            name_of_file =" letter: " + str(index)+ ".png"
            OpenCV.saveImg(inp_image[x:x+w, y:y+h], name_of_file)
            list_of_coors.append([x,y,w,h])
        Endif
    next index
    endfor
    return list_of_coors //this will be used later for sorting letters into lines and words using their coordinates and later in the GUI to output bounding boxes.
endFunction
```

- Make letter contours thicker:

  For every letter image (.PNG) saved using function **detect_Letter():**

  1) Format the image into a grayscale image. Save it as **imgGray**
  2) (This is optional) You can use Binary Inverse Thresholding on the grayscale image.
  3) Use findContours() function from OpenCV to detect the contours of the letter in the image. Save this array as **contour** (it should have only 1 index)
  4) Use drawContours() function from OpenCV to draw over the contours of the letter. You can specify the thickness of the drawing (by drawContours()) by using the parameter thickness =

[number]. Use this parameter to make a thick outline over the letter.

5) Save this new image as a .PNG file and use this for recognition.

Pseudocode:

```
procedure thickness_enlarge(num_of_contour_images):
    for index = 0 to num_of_contour_images-1
        name_of_file = “letter: ” + str(index) + “.png”
        inp_image = OpenCV.open(name_of_file)
        imgGray = OpenCV.cvtColorGray(inp_image)
        contour = OpenCV.findContours(ingGray)
        imgThick = OpenCV.drawContours(imgGray, contour, thickness = 10)
        name_1 = “thick: ” + str(index) + “.png”
        OpenCV.saveImg(imgThick, name1)
    next index
endProcedure
```

- Partition letters into lines/rows:
    1) Call the detect_Letter function which returns coordinates of each letter (along with width and height) in an array. Assign this array to a variable called list_coor.
    2) Assign an empty list to the global variable list_coors_with_label
    3) For each index of list_coor
        1) Make a temporary list with only two indexes [list_coor[x] and index number]
        2) Append this new temporary list to array list_coors_with_label

        //We give each set of coordinates a label which corresponds to the letter image (index number) which we created in detect_Letter() and thickness_enlarge() so that when we do bubble sort the relative position of each set of coordinates is not lost (in relation to the images of letters).

    4) Make a bubble sort algorithm function to sort out the list list_coors_with_label in order of y axis coordinates of each index. Call the sorted list sorted_list_coors.

    5) Make a global variable called temp

    6) Make an empty array called list_coors_with_rows with a global scope

7) For every index in sorted_list_coors:
    1) If the difference between y-axis coordinate in the current index and the y-axis coordinate in the adjacent index is bigger than a certain threshold (e.g. 100):
        1) This tells us that a new row starts here in the input writing. Append sorted_list_coors[temp: x+1] (where x is the index), this is the coordinates of letters in one row
        2) Set temp to x+1
8) Return sorted_list_coors

Pseodocode:

```
function bubble_sort_row(list_coors)
    while True
        swap = False
        for x = 0 to len(list_coors)-1:
            if list_coors[x][0][1] > list_coors[x+1][0][1] then
                swap = True
                temp = list_coors[x]
                list_coors[x] = list_coors[x+1]
                list_coors[x+1] = temp
            endif
        next x
        if swap == False:
            return list_coors
        endif
    endwhile
endfunction


function organise_row(inp_image)
    list_coors = detect_Letter(inp_image)
    list_coors_with_label = []
    for index = 0 to len(list_coors)-1
        temp_array = [list_coors[index], index]
        list_coors_with_label.append(temp_array)
```

```
        next index
        sorted_list_coors = bubble_sort_row(list_coors_with_label):
        list_coors_with_rows = []
        temp_for_list = 0
        for index = 0 to len(sorted_list_coors) -1:
            if (abs(sorted_list_coors[index][0][1] – sorted_list_coors[index + 1][0][1])>100) then
                new_row = sorted_list_coors[temp: x + 1]
                list_coors_with_rows.append(new_row)
                temp = x+1
            endif
        next index
        return list_coors_with_rows
endFunction
```

Update 20/03/2022: The pseudocode marked with the red outline is wrong. During development, I tried this algorithm out and I had to make an amendment for it to run as intended. The new pseudocode to replace that is:

```
    for index = 0 to len(sorted_list_coors) -1:
        if (abs(sorted_list_coors[index][0][1] – sorted_list_coors[index + 1][0][1])>100) then
            new_row = sorted_list_coors[temp: x + 1]
            list_coors_with_rows.append(new_row)
            temp = x+1
        if index == len(sorted_list_coors)-2:
            new_row = sorted_list_coors [temp: x+2]
            list_coors_with_rows.append(new_row)
```

- Group letters to form words:
    1) Call the function organise_row(), and save its output as list_coors_in_rows.
    2) Make an empty array called list_coors_in_row_with_words.
    3) For every element in list_coors_in_rows (for loop with var x) :
        1) Do bubble sort on list_coors_in_rows[x] and assign the list as row_organised.
        2) Make an empty array called words_list:
        3) Make a for loop again (with var y), this time make a loop that iterates through elements in row_organised [x]
            1) Assign a variable, temp, to 0:

2) If row_organised[y+1][0][0] – row_organised[y][0][0] > 50
    1) Append row_organised[temp: y+1] to words_list
    2) Temp = x+1

3)Append words_list to list_coors_in_row_with_words.

4) Return list_coors_in_row_with_words.

Pseodocode:

```
Function bubble_sort_with_x_axis(list_coors):
        while True:
                swap = False
                for x = 0 to len(list_coors)-1:
                        if list_coors[x][0][0] > list_coors[x+1][0][0] then
                                swap = True
                                temp = list_coors[x]
                                list_coors[x] = list_coors[x+1]
                                list_coors[x+1] = temp
                        endif
                next x
        endwhile
        return list_coors
endfunction
Function group_letters_to_form_words(inp_image):
        list_coors_in_rows = organise_row(inp_image)
        list_coors_in_row_with_words = []
        for x = 0 to len(list_coors_in_rows)-1:
                row_organised = bubble_sort_with_x_axis(list_coors_in_rows[x])
                words_list = []
                for index = 0 to len(row_organised)-1
                        temp = 0
                        if abs(row_organised[index+1][0][0] – row_organised[index][0][0]) > 50 then
                                words_list.append(row_organised[temp: x+1])
                                temp = x+1
```

```
                endif
            next index
            list_coors_in_row_with_words.append(words_list)
        next x
        return list_coors_in_row_with_words
endfunction
```

Section 2.2.3) Graphical User Interface:

Graphical User Interface is one of the most important aspect in software development, because it is the bridge between the functionality of a software and the user (for whom it is designed). I want to design a user interface where the user does not feel overwhelmed or lost when using the interface. Therefore, the user interface should be simple, aesthetically clean (with an even colour scheme) and accessible with instructions. In the Product Analysis, one of the reason why I favoured Google Lens over Pen to Print because Google Lens gave the user clear instructions and had a great widget placement (such as swiping to accessing Text recognision features, etc). Whereas Pen to Print gave a basic user interface where there were little to no assistance.

I hope to use features such as placing bounding boxes around text regions, and cropping each line of input and displaying its generated text below it, etc. The overview for this section is given below:

For this section, I will be using the Tkinter library of Python because this library contains all the necessary widgets and canvases I need for my Graphical user interface.

[Note: I will not be able to write about every objective in the Graphical User Interface section in the "Key features of my solution" because some of them are aesthetic, or structural aspects that cannot be accurately represented as pseudocode]

1) Add a zoom in and out operations for image for user to view: I will Tkinter slider widget which allows the user to slide a widget that I can use to determine my zooming scale factor. I can use basic division rules in maths to output a new image that is reduced/enlarged in size (depends on user slider input).
2) Show bounding boxes around each detected letter on the input image: I will call the function **detect_Letter()** (that I made in the "Image Recognition" section), and use its output (which is an array of coordinates of where each letter is placed on the input) to make transparent boxes with a colour outline around each letter on the input.
3) Show snippet of each row of input with text output beneath it: I will use the function **organise_row**() (that I made in the "Image Recognition" section),to find the coordinates of the start and end of each row. Then using NumPy slicing, I can crop each line of input and plot it on a separate window. I can use the the Tkinter Entry widget to output generated text for that line alongside allowing the user to amend it.
4) Make an output file: Using the output of function **group_letters_to_form_words()** (that I made in the "Image Recognition" section), I can write/overwrite (existing files) of text document and output the path to that file.

My project: Handwriting recognition research project

Detailed breakdown of this section:

1) Add a zoom in and out operations for image for user to view:
    1) Define a Tkinter Slider method(). Assign it to the variable **slider_input**
    2) Store the width and height of the image in an ar**ray dimensions.**
    3) Define a new variable **scale_factor**, assign it to slider_input/100
    4) Multiply the 2 indexes of **dimensions,** by **scale_factor**, and save it as the new width and height of the image.
    5) Update the Tkinter image widget with the new image
    6) Return the input_image

Pseodocode:

```
Function zoom_in_out(input_image)
        slider_input = Tkinter.Slider(min = 0, max = 200, preset = 100)
        slider_input.grid()
        dimensions = [input_image.width(), input_image.height()]
        scale_factor = slider_input.get()/100
        new_dimensions = [input_image.width()*scale_factor, input_image.height()*scale_factor]
        input_image.width() = new_dimensions[0]
        input_image.height() = new_dimensions[1]
        canvas.update(image = input_image)
        return input_image
endFunction
```

2) Show bounding boxes around each detected letter on the input image:
    1) Call the function **detect_Letter()**, assign its output (list of coordinates of where letters are on the input (x, y, width, height (of each object))) to **bounding_coors.**
    2) For each index in **bounding_coors:**
        1) Use a Tkinter method to make output rectangle with start coordinates (x, y) and end coordinates(x+w, y+h) on the image.

Pseodocode:

```
Function bounding_boxes(input_image):
        bounding_coors = detect_Letter(input_image)
        for index = 0 to len(bounding_coors)-1:
                x = bounding_coors[x][0]
                y = bounding_coors[x][1]
                w = bounding_coors[x][2]
```

```
            h = bounding_coors[x][3]

            Tkinter.makeRectangle(input_image, (x, y), (x + w, y + h))

        next index

        return input_image

endfunction
```

3)Show snippet of each row of input with text output beneath it:

1) Call the organise_row(), assign its output (list of letter coordinates that are separated into rows) to row_list
2) Use the bubble sort algorithm of row_list where you compare the x-axis coordinates of adjacent indexes in each row (indexes of row_list). Call this variable organised_row_list.
3) Make an empty array crop_region_each_row.
4) For each index in organised_row_list:
    1) Save the (x,y) coordinates of the the first index, and the (x+w, y+h) in the last index in the variable temp_crop_array.
    2) Append it to the array crop_region_each_row.
5) For every index in crop_region_each_row:
    1) Use numpy slicing between the coordinates in crop_region_each_row[index], which will crop the individual row of the image
    2) Save this as a .PNG file which can be later retrieved for the user interface.

Pseodocode:

```
Function bubbleSortInRow(list_with_rows)

        while True

            swap = False

            for row = 0 to len(list_with_rows)-1

                for index = 0 to len(list_with_rows[row])-1


                                if list_with_rows[row][index][0][0] >
list_with_rows[row][index+1][0][0] then

                                    swap = True

                                    temp = list_with_rows[row][index]

                                    list_with_rows[row][index] = list_with_rows[row][index + 1]

                                    list_with_rows[row][index + 1] = temp
```

```
                    endif
                next index
            next row
        return list_with_rows
endFunction
Function snippet_rows(input_image)
        crop_region_each_row = []
        row_list = organise_row(input_image)
        organised_row_list = bubbleSortInRow(row_list)
        for index = 0 to  len(organised_row_list)-1:
            start = [organised_row_list[index][0][0][0], organised_row_list[index][0][0][1]]
            end = [organised_row_list[index][-1][0][0] + organised_row_list[index][-1][0][2],
organised_row_list[index][-1][0][1] + organised_row_list[index][-1][0][3]]
            temp_crop_array = [start, end]
            crop_region_each_row.append(temp_crop_array)
        next index
        for index = 0 to len(crop_region_each_row)-1:
            name_of_file = "row" + str(index + 1) + ".PNG"
            crop_row_image =
OpenCV.saveImg(input_image[crop_region_each_row[index][0][0]:
crop_region_each_row[index][1][0]], crop_region_each_row[index][0][1]:
crop_region_each_row[index][1][1]])
        next index
endFunction
```

4) Make an output file:

1) Call the function final_Output(), save the output of the function as text_in_array.

2) Create a new file named "final_output.txt".

3) Write/Overwrite each index of text_in_array as a new line.

Pseodocode:

```
procedure output_file()
        text_in_array = final_Output()
        file_Open = openWrite("final_output.txt")
```

```
    for x = 0 to len(text_in_array)-1:
        file_Open.writeLine(text_in_array)
    next x
    file_Open.close()
endProcedure
```

[Flowchart how user interface should be navigated]

Section 2.3) Detailed Design of the Graphical User Interface

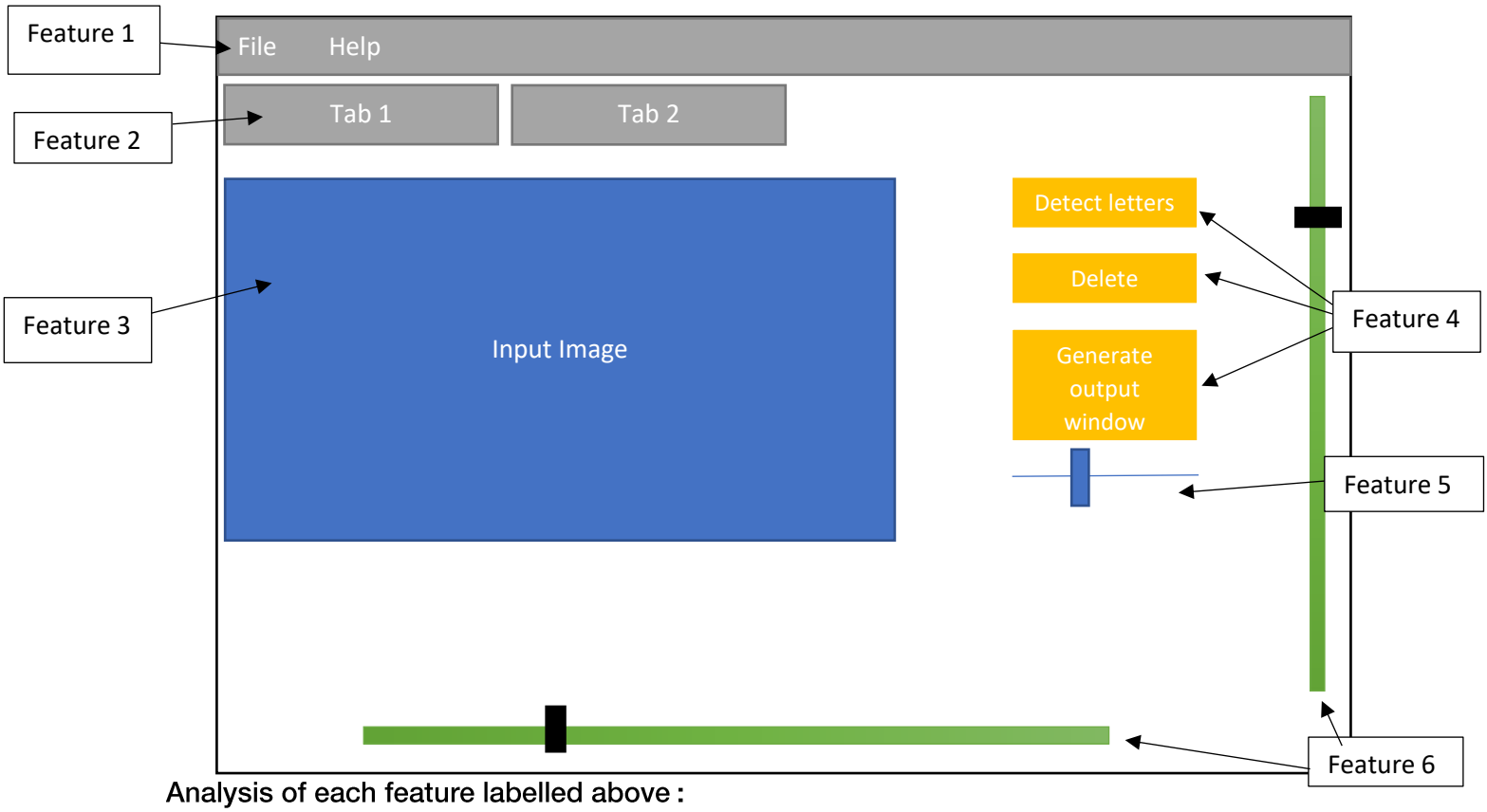
1) Inputs and outputs:
   My graphical user interface design can be split into inputs and outputs (I will account for aesthetics and widget position later in this section).
   The parts of my program that will require user interface are:
   a) Asking for image file containing handwritten text (however only .PNG, and .JPEG/.JPG are allowed, because of the timeframe of completing the project I don't think I can account for other file types in my project).
   b) Text entry: This will output generated text for each row (output from my CNN). However, the user can change any spelling mistakes that were made in my output
   c) Button clicks:
      1) Detecting text button (in "Home window"): This is the button that the user can press if they want to check what text is detected.
      2) Recognition button (in "Home window"): This is the button that will generate a separate output window that will contain digitally generated text (which can be amended by the user).
      3) Delete button (in "Home window"): This is to remove the bounding boxes from the image canvas (in Tkinter).
      4) Generate Output File (in "Output window"): This will generate an output text file, based on the text entries in "Output Windows".
   d) Zoom in/out: This is a slider that the user can use to increase/decrease the image size while viewing. This is useful when using bounding boxes are shown, so the user can view exactly what text is detected.

   The outputs in my program will be:

   a) Output window: The main output of my program will be a separate output window, that will output a cropped image of each input row and its corresponding generated output text underneath.
   b) Output file that contains each row of generated text.

2) Aesthetic look of the user interface:
   One of the most parts of my user interface is the look and feel of it. I want to design a user interface that does not look complicated or overwhelming, instead I want to make a user interface that looks clean and condensed with features that are highlighted and showcased properly.
   This is my initial draft of the first window ("Home page") of my program

My project: Handwriting recognition research project



Analysis of each feature labelled above :

Feature 1: This is a menu bar. When the user clicks on this menu bar, they can access functionalities such as opening a new file, or going on the help window, which will provide additional information.

Feature 2: In this program I have incorporated a feature to have tabbed windows. This means that the multiple windows in my program (such as Output window or Help window) can be accessed by hovering and clicking the tabs above instead of having to open a new instance each time. This is incredibly useful for ease of use because everything in my program will be accessible to the user without being overwhelmed or complicated.

Feature 3: This is where the input image (by the user) will be shown. This is the image on which several operations will be done such as zooming in/out and placing bounding boxes.

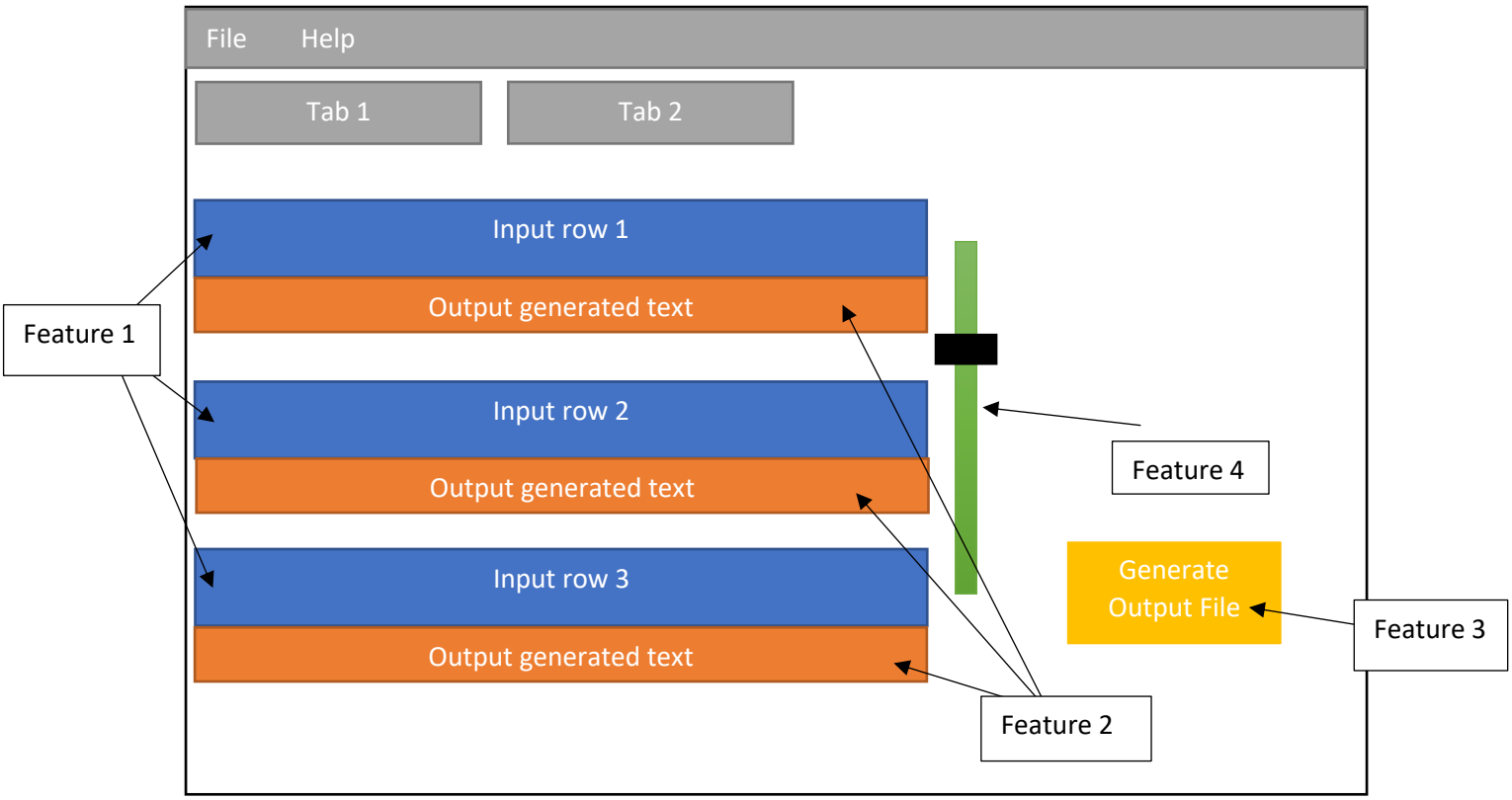
Feature 4: These are buttons for my user interface:

1) Detect Letters: This will output bounding boxes over the input image (**Feature 3**) so the user can view what regions of text have been detected in the input.
2) Delete: This will delete the bounding boxes that are over the input image.
3) Generate output window: If the user presses this button, they will be redirected to a new tabbed window where they will receive a digtally generated output (using my CNN)

Feature 5: This is the zoom slider that the user can use to zoom in and out the input image

Feature 6: These are horizontal and vertical scrollbars, that can be used to navigate and view unreachable image data (due to image widget size constraint) of the input image.

This is a draft of my second window ("The Output window")



This is a draft of my second window ("The Output window"):

Feature 1: This is the cropped images of each individual lines of input text paragraphs from the input image.

Feature 2: This is the generated text for the line of text (from input) directly above. It's a Tkinter Entry box meaning it can be amended by the user.

Feature 3: This is a button. If it is pressed, then an output file (containing the text from the Entry box in Feature 2) is created.

Feature 4: This is a scrollbar that can be used to navigate the generated texts and cropped input rows (if there are more than 3).

Section 2.4) Usability Features:

I designed my user interface so that any kind of user can use it (not just a particular type or demographic).

In my "Home" and "Output" windows, I have ensured that my widgets to show input images and individual row images have large dimensions (with respect to the dimensions of the window) so the scope of the image is not too small for the user to view (especially when bounding boxes are output). Also, I have assigned scrollbars in

both windows, so the user is able to access all the parts of their input data and all the rows of their image.

I have also designed tabbed windows in the user interface, so all the tabs are easily accessible all at once to the user. This is much preferable than opening individual windows which would require the user to access the taskbar (in Windows OS) to view each tab, which may be less intuitive.

I have also incorporated zoom functionality for viewing the input image in the "Home" window. This will simplify navigating extremely small or extremely big images in my program.

In terms of aesthetics of the user interface, I will keep a consistent and plain colour scheme on all my windows.

I have tried to implement large buttons with clear text instructions with sufficient equal spacing, so the user doesn't encounter any problems such as clicking the wrong button or not understanding what a button does (due to poor instructions).

Section 2.5) Key Variables/data structures/classes:

Name	Variable/ data structure/ class/function	Explanation/Justification
Network()	**Class**	This is the most important part of my project because the Network() class is the only class responsible for taking input data (letters) and outputting a recognition result. Without Network() my program solution would have no output which would render the whole concept useless.
Backpropagation()	**Method of Network**	Backpropagation is a part of the Network() class, however it is the most important aspect of neural networks. Backpropagation is the only method responsible for "learning" from past errors and updating weights and biases so it is more accurate in

		predicting new data. Without backpropagation(), my program would just give out erroneous outputs.
feedforward()	**Procedure of Network**	This is a procedure that is responsible for performing activation (sigmoid and softmax) on input data and weights and biases so that it can be used in backpropagation. While it is not directly important in the project, but it plays a pivotal role in the neural network because without feedforward, all the input data cannot be decomposed and analysed for updating the neural network and outputting a result for input.
biases_layer2, biases_layer3	**Attribute of Network**	These are the biases of layers 2 and 3 (respectively) in my neural network. These biases will be used in my feedforward() for outputting the results of sigmoid and softmax activations. Without biases, the neural network would be inaccurate and the training of neural network would be largely unreliable.
weights_layer2, weights_layer3	**Attribute of Network**	These are the weights of layers 2 and 3 (respectively). The purpose of these weights are to be multiplied with input data and use it in deducing the output of the network. Without weights, my neural network would simply not

		work, because there is no other basis for my neural network to compare input data against and come up with an output.
activation_sigmoid	**Attribute of Network**	This is an attribute of Network that stores the outputs (array) of sigmoid activation. This will be combined with weights of layer 3 and will be used for output for softmax. Activation_sigmoid is also used in backpropagation. Therefore, activation_sigmoid is very important.
activation_softmax	**Attribute of Network**	This is an attribute of Network that stores the output of softmax activation (array). This is also the output of the network for an input. The element containing the highest probability is the node that is the output. Without activation_softmax, I would have no output for my neural network
dWeights2, dWeights3, dBias2, dBias3	**Variables in backpropagation()**	These are the variables that store values for how much weights and biases of the neural network are to be “nudged” by. dWeights2, dWeights3, dBias2 and dBias3 are stored in backpropagation() and are its output. These variables update the network parameters so that the neural network can “learn” for the next iteration.
detect_Letter()	**Function**	detect_Letter() is the function responsible for detecting letters in any

		handwritten input data and output a standalone image path for the detected letters so they can be used as inputs for Network(). Without detect_Letter(), none of my user data can be processed and so there would be no output for the user.

Section 2.6) Validation

Validation of user input can be used to create a robust and error-free program. Therefore, I need to deduce what type of data is accepted for my inputs and output necessary instructions if the input is invalid.

I have already defined the inputs of my user interface in Section 2.3, this is how they will be validated:

1) Buttons: None of the buttons need any major validation because it is uses Tkinter methods that usually work errorfree (given that there are no mistakes in my methods that I link to the button).

   The only validations I will need to do are:

   1) Check whether an input image is provided by the user, before the buttons (Detect letters, Delete, and/or Generate output window in "Home window") are pressed. This is because, if it isn't validated, my image processing, and neural network algorithms may try to use empty/null image data, which will output an error.
   2) Make sure the Delete button cannot be used before the Detecting Letters button otherwise my algorithm for deleting bounding boxes may fail as there would be no bounding boxes as input for the algorithm.

2) Asking for image file: Make sure the image file is a .PNG or .JPG/.JPEG, so that my algorithms don't potentially malfunction, especially in my image processing where I explicitly accept only .JPEG or .PNG

3) Text entry: There is no validation I can do here because the user input will not directly cause any errors for my program.

4) Zoom in/out: Since this is a slider, the only input it can accept is numerical data between a given range (in my case between 0 and 200), meaning there is no need for me to validate the output of the Zoom function.

Section 2.7) Testing (for development)

Section 2.7.1) Neural Network:

My project: Handwriting recognition research project

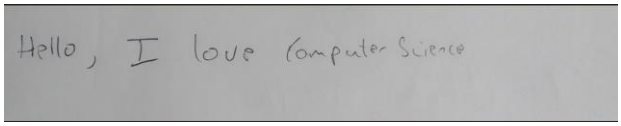
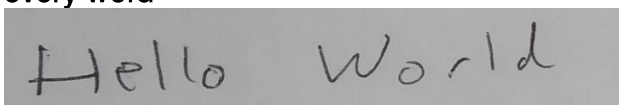
The type of input I will accept for testing my neural network is .JPEG/.JPG or .PNG, where an individual letter is the foreground.

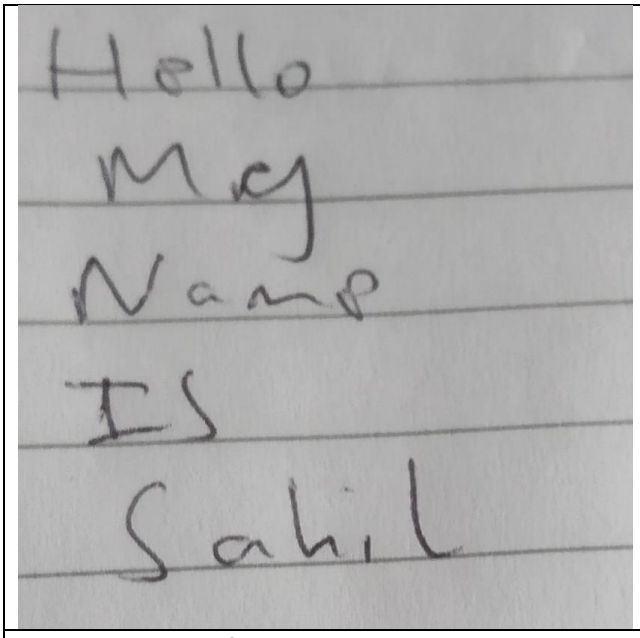
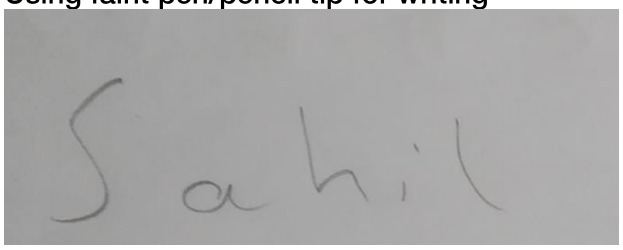
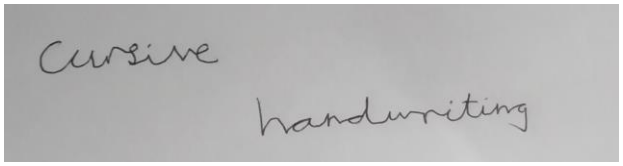
Test data	Valid/Boundary/Erroneous	Explanation/Justification
Image with no noise 	Valid	This image has no noise, meaning the only black pixels in the image are that of the actual letter. While the complete accuracy of the letter being recognised is not assured, this image does assure high accuracy.
Properly drawn letter 	Valid	If a letter is drawn out neatly and properly, it makes it easier for the neural network to recognise the letter as it is most likely to match the EMNIST dataset images of the same letter.
Drawing the letter at an angle 	Boundary	Drawing the letter at an angle can make it harder for the neural network to recognise the letter as it is less likely to match EMNIST dataset images of the same letter (which are probably not rotated). However, the shape of the letter is identical to the EMNIST dataset images, so the letter can still be recognised (but with less accuracy).
Noise 	Erroneous	The neural network feeds in black pixels (assuming white background) to recognise a letter. If the image contains noise, it too is sent to the neural network which can and probably will skew up the final output.
Letter not drawn correctly 	Erroneous	If the letter is drawn wrongly, it can't be compared properly against the EMNIST dataset images, and therefore the output will most likely be false.

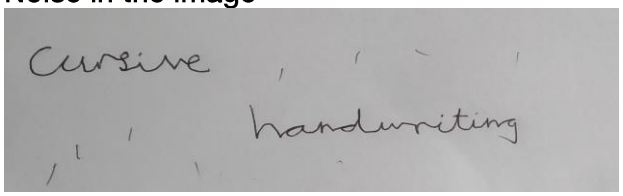
Section 2.7.2) Image Processing:

My project: Handwriting recognition research project

1) Detecting Letters: The input data I will accept is a .JPEG/.JPG or .PNG with handwritten text. This will be user input in my final user interface.

Test data	Valid/Boundary/ Erroneous	Explanation/Justification
Plain white background, handwriting written in black ink. 	Valid	If the input image has a white background and black ink for handwriting, it will highlight the handwritten text better for neural network recognition than any other background and foreground combination.
Sufficient spacing between every letter in every word 	Valid	My algorithm for image processing uses Contour detection from the OpenCV library. For it to recognise each individual letter in the image, there should be sufficient spacing between each adjacent letters otherwise two or more letters can be detected as one object which will skew up my recognition outputs for CNN.
Using lined paper instead of plain paper	Boundary	Since my image processing algorithm uses Contour detection from the OpenCV library, it doesn't explicitly detect letters, if the image has lines, diagrams, noise, etc. it will detect that and feed it forward to the neural network. However, paper lines are faint, and

		hence it might be erased during denoising the image.
Using faint pen/pencil tip for writing 	Boundary	It depends on how the lighting conditions are in the input image. If the paper is scanned in through a printer (where lighting is even, and any external noise is eliminated) faint pencil lines will probably be detected. Otherwise, it cannot be guaranteed.
Using cursive handwriting instead of block text 	Erroneous	In cursive handwriting, all the letters are joint up to form a word. However, since there are no gaps between each letter, my image processing algorithm cannot detect each individual letter in the image. This will produce unreliable results as these detected objects will be feedforward to the CNN, which

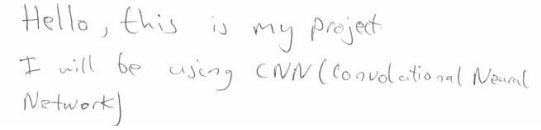
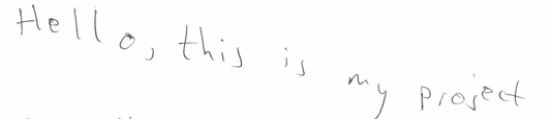
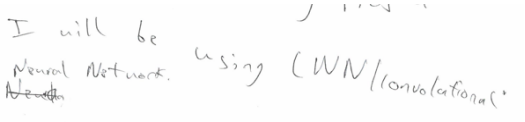
		will produce the necessary output.
Noise in the image 	Erroneous	Since my image processing algorithm uses Contour detection from the OpenCV library, it doesn't explicitly detect letters, it will detect any noise in the image as well. This will produce erroneous text outputs as these noise images will be feed into the CNN which will produce nonsensical outputs.

2) Thickness of the letter contours enlarged: The type of input I will accept is .JPEG/.JPG or .PNG, where an individual letter is the foreground.

Test data	Valid/Boundary/Erroneous	Explanation/Justification
Letter image with no noise 	Valid	This image has no noise, meaning the only black pixels in the image are that of the actual letter. This means only the letter contours are made thicker and highlighted, instead of noise. This will provide a clean image for the neural network to recognise.
Image only contains one letter 	Valid	This image only contains one letter, so only the letter contour is made thicker and highlighted.
Part of letter is cut during automated cropping 	Boundary	If a small proportion of the letter is cut during letter cropping, the neural network may still be able to recognise the letter but the accuracy of it will decrease.
Noise in the image 	Erroneous	Since my algorithm uses Contour detection from OpenCV, it doesn't specifically only detect

		letters, therefore any noise that is detected will be enlarged as well. This will ruin the image and the neural network output will be skewed.

3) Partition detected letters into rows: The input data I will accept is a .JPEG/.JPG or .PNG with handwritten text.

Test data	Valid/Boundary/Erroneous	Explanation/Justification
Each row of words is in a straight line (the y-axis of each letter in each row is equal (+- some error interval)) 	Valid	My algorithm relies on each letter having equal (+- some error interval) y-axis coordinates with other letters and encountering a significant step between the next line. Therefore, if the row is written straight, then row sorting would be easier and more accurate with my program.
Row of words is slanted (human error) 	Boundary	Since humans are not perfect at writing in straight lines (especially on plain paper), this is a problem I may encounter. If the row is not too slanted (i.e. the difference between y-axis of letters is not outside error interval) then my program should work as planned.
Row of line is too slanted 	Erroneous	If the line is too slanted (i.e. the difference between y-axis of letters is outside error interval) then some letters would be grouped in a new line, which will be erroneous.

4) Group detected letters to form words: This will take data that has one row of handwritten data (.JPEG/.JPG or .PNG).

Test data	Valid/Boundary/ Erroneous	Explanation
The gap between each word is more than gaps between each letter in a word. 	Valid	Since my algorithm involves comparing the length between each adjacent letter to a certain threshold (if it is bigger then it is deduced that a new word is started).
The gaps between each letter are a bit big 	Boundary	Since the gaps between each letter is big, there is uncertainty whether my program will correctly classify words properly using coordinates.
The gaps between each letter are too big 	Erroneous	This will definitely produce erroneous results as the gaps between each letter in each word is too large and my program will class each letter as a new word.

Section 2.8) Post Development data

Post Development testing:

This is my post development testing for my project:

Test data	Valid/Boundary/Erroneous	Explanation/Justification
Input a .JPEG/JPG/PNG file containing handwritten data on a plain white background, and output a text file containing the converted digital text	Valid	This is the main purpose of my program, it should accept user input and output a suitable text file containing text outputs based on user input. If my program cannot fulfil this objective, it has failed to be a solution to my problem.
Input a .JPEG/JPG/PNG file containing handwritten data on a lined paper background, and output a text file containing the converted digital text	Boundary	With my prior knowledge of OpenCV and letter detection, I know that eliminating backgrounds such as lined paper is difficult, instead it will likely create problems during letter detection. However,

		lined paper is what most people use, and they may use it for my program, so I need to verify whether my program can accept this or not.
Input a .JPEG/JPG/PNG file containing cursive hanwrighting, and output a text file containing the converted digital text	Erroneous	I know from prior experience that contour detection in OpenCV will not accept cursive handwriting (due to the fact that all letters in a word are joint up and contour detection requires some sort of spacing between each contour, therefore the letter will not be individually detected). Therefore I know that using cursive handwritten input will create erroneous outputs.

Section 3) Development section

Section 3.1) Stage 1 of development (neural network made from scratch)

Section 3.1.1) Preparing the MNIST/EMNIST dataset for neural network

I have an algorithm for my neural network (Design Section), however, to test my algorithm, I will need test data, in this case the EMNIST dataset. However, to implement the EMNIST dataset, I would need some way to unpack and rearrange the dataset elements so it can accommodate my neural network.

For the initial stage of my neural network development, I will be using the MNIST (digits between 0-9) dataset because it is a smaller dataset than EMNIST, which is helpful because it means that loading and unpacking the dataset will be faster, and the average neural network training execution will be faster than if I used EMNIST. Also, since using MNIST means there are only 10 possible outputs, I will only have to worry about 10 output nodes during development. This will reduce the complexity of developing the program significantly. If my neural network algorithm works with MNIST dataset, I can easily just substitute my training neural network data with EMNIST and change the output nodes accordingly.

Note: Both MNIST and EMNIST contain 28x28 greyscale images of their respective digits/letters, which is another reason for me using MNIST for initial development, because there are minimal transformations (if any) that I would have to do to substitute MNIST with EMNIST.

My project: Handwriting recognition research project

For my neural network, I want my nodes' array dimensions as specified below:

Layer	Array Dimensions	Explanation
Layer 1 (input layer)	(1, 784)	Since there are 784 pixels for each image in the MNIST (and EMNIST) dataset, I will need to have 784 nodes where each node represents each pixel value.
Layer 2 (hidden layer)	(10, 784)	There will be 10 nodes in Layer 2 (this can be changed in the future for experimentation), each containing a list with 784 notes (so I can carry out vector dot product (which would require equal dimensions between input array layer, and hidden array layer)).
Layer 3 (output with applied softmax activation)	(10, 10)	I will be applying the softmax algorithm using Layer 3 weights. The array of values that will be output by using softmax will be used to output a prediction for the neural network input data.

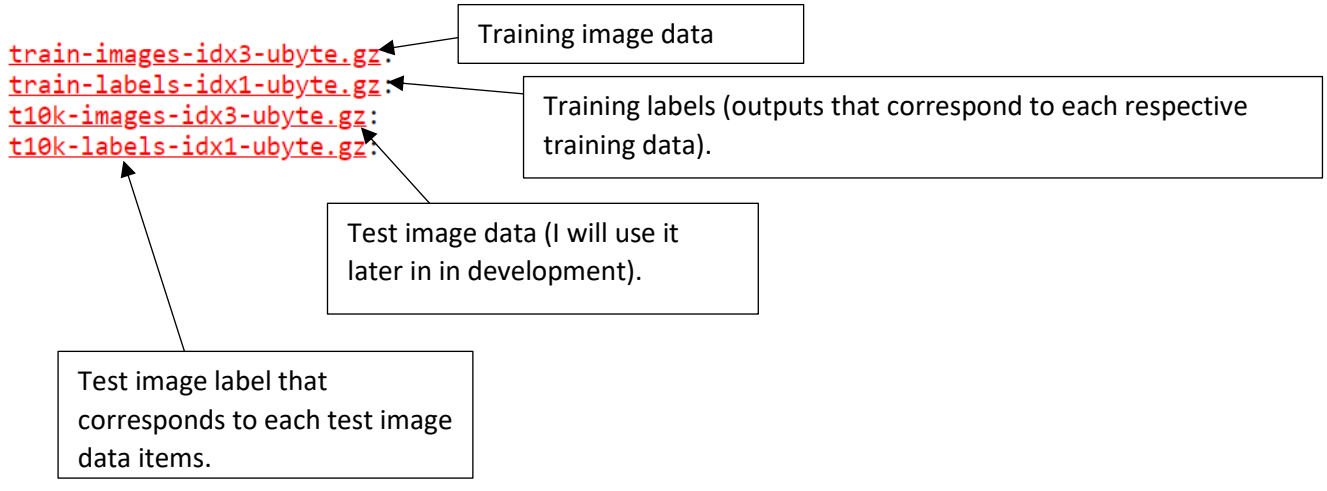
Looking at the table above, I need to make sure that my MNIST training data array needs to be the same size as the input layer array.

MNISTtrainingData = [numpyArray, numpyArray, numpyArray……,numpyArray] (size of MNISTtrainingData will be the number of training images I will have).

Each NumPy Array (numpyArray) in MNISTtrainingData will be of dimension size (1, 784) (the size of input layer array).

First of all, I need to download the MNIST dataset from online, so I will download it from MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.

The 4 files I downloaded:

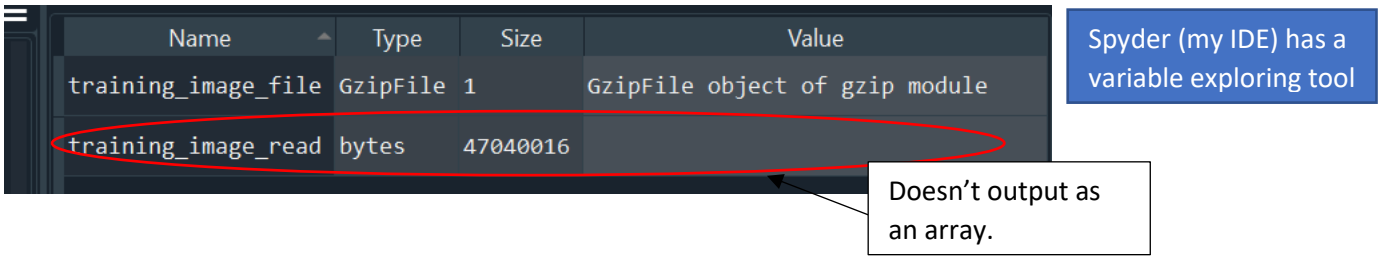


My project: Handwriting recognition research project

Since this gzip file, I need to uncompress it. To do this, I will use Python gzip library.

```
import gzip
training_image_file = gzip.open("train-images-idx3-ubyte.gz", "rb")#finds and opens local file directory of training images
training_image_read = training_image_file.read() #reads the contents of training_image_file
training_image_file.close()#closes file
```

However, I realised that the gzip read() method doesn't actually load the training dataset array. The value of training_image_read is not an array (as you can see below).



After researching online on how to extract NumPy arrays from a IDX file format, I found this Stack Overflow answers (mnist - Extract images from .idx3-ubyte file or GZIP via Python - Stack Overflow), where they import the library idx2numpy, and use convert_from_file() method from that library. This outputs a NumPy array.

Therefore, I added the code below, in which the NumPy array containing the dataset will be assigned the variable **training_data_read**.

```
import idx2numpy
training_data_read = idx2numpy.convert_from_file(training_image_file)
```

However, this time when I execute my Python program, I receive an Value Error.

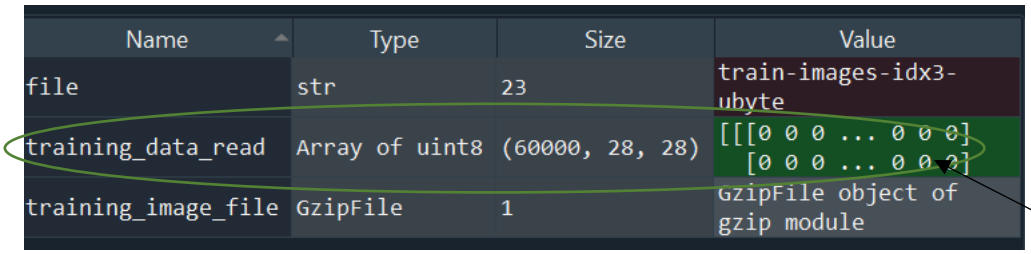
```
  File "C:\Users\Sahil\anaconda3\lib\_compression.py", line 14,
in _check_not_closed
    raise ValueError("I/O operation on closed file")

ValueError: I/O operation on closed file
```

Since, the error involves "closing a file", I figured out that I need to remove **training_image_file.close()** from its place and move it down beneath the line **training_data_read = idx2numpy.convert_from_file(training_image_file)**. I additionally had to remove the **training_image_read = training_image_file.read()** because it is not really needed and may cause further errors, because it involves accessing the gzip file just before using idx2numpy. This is my code after amending it:

```
import gzip
training_image_file = gzip.open("train-images-idx3-ubyte.gz", "rb")#finds and opens local file directory of training images
import idx2numpy
training_data_read = idx2numpy.convert_from_file(training_image_file)#reads from the IDX file and will output a NumPy array
training_image_file.close()
```

My project: Handwriting recognition research project

During this run, I have managed to get a NumPy array as **training_data_read.** The array has dimensions (6000, 28, 28), meaning that the array has 60000 elements, which are 2-D arrays with dimensions (28, 28) where each dimension represents the row and column respectively.



Name	Type	Size	Value
file	str	23	train-images-idx3-ubyte
training_data_read	Array of uint8	(60000, 28, 28)	[[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
training_image_file	GzipFile	1	GzipFile object of gzip module

Spyder (my IDE) has a variable exploring tool

Array containing all 60000 MNIST images.

I repeated the same process for the files **"train-labels-idx1-ubyte.gz", "t10k-images-idx3-ubyte.gz", "t10k-labels-idx1-ubyte.gz"** which contain the training data labels, test image data, and test image labels respectively.

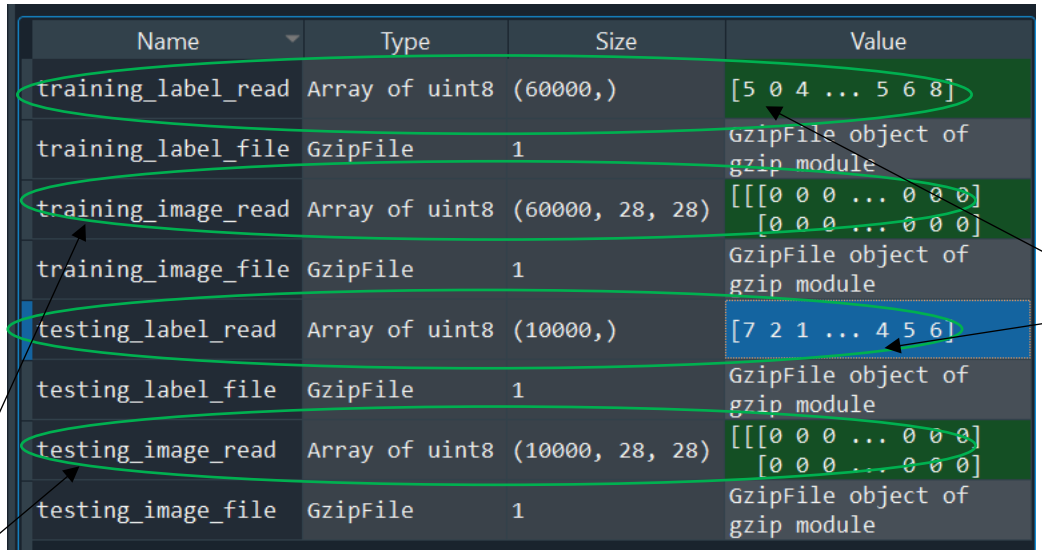
```
import gzip
training_image_file = gzip.open("train-images-idx3-ubyte.gz", "rb")#finds and opens local file directory of training images
training_label_file = gzip.open("train-labels-idx1-ubyte.gz", "rb")#finds and opens local file directory of training labels
testing_image_file = gzip.open("t10k-images-idx3-ubyte.gz", "rb")#finds and opens local file directory of testing images
testing_label_file = gzip.open("t10k-labels-idx1-ubyte.gz", "rb")#finds and opens local file directory of testing labels
import idx2numpy
training_image_read = idx2numpy.convert_from_file(training_image_file) #reads from the training images IDX file and will output a NumPy array
training_label_read = idx2numpy.convert_from_file(training_label_file) #reads from the training labels IDX file and will output a NumPy array
testing_image_read = idx2numpy.convert_from_file(testing_image_file) #reads from the testing images IDX file and will output a NumPy array
testing_label_read = idx2numpy.convert_from_file(testing_label_file) #reads from the testing labels IDX file and will output a NumPy array
training_image_file.close() # closing training images file
training_label_file.close() # closing training labels file
testing_image_file.close() # closing testing image file
testing_label_file.close() # closing testing labels file
```

Loading gzip files

Converting to NumPy

Closing gzip files

What do the variables **training_image_read, training_label_read, testing_image_read, testing_label_read** contain?



Name	Type	Size	Value
training_label_read	Array of uint8	(60000,)	[5 0 4 ... 5 6 8]
training_label_file	GzipFile	1	GzipFile object of gzip module
training_image_read	Array of uint8	(60000, 28, 28)	[[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
training_image_file	GzipFile	1	GzipFile object of gzip module
testing_label_read	Array of uint8	(10000,)	[7 2 1 ... 4 5 6]
testing_label_file	GzipFile	1	GzipFile object of gzip module
testing_image_read	Array of uint8	(10000, 28, 28)	[[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
testing_image_file	GzipFile	1	GzipFile object of gzip module

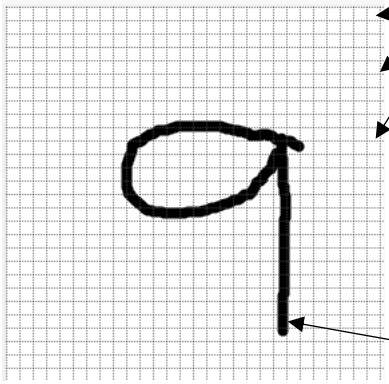
Spyder's variable exploring tool

training_label_read and **testing_label_read** is a one-dimensional array containing a list of number outputs that correspond to image data **training_image_read**, and **testing_image_read**.

training_image_read and **testing_image_read** have the similar array structure. **testing_image_read** has 10000 elements, whereas **training_image_read** 60000 elements, but each element in each array has a two-dimensional array of dimensions (28,28) where both 28s represent rows and columns of the image.

My project: Handwriting recognition research project

Visualizing training_image_read and testing_image_read:



training_image_read[start:end][x] and testing_image_read[start:end][x] outputs the array of pixels for row x+1 of each image (between indexes start and end) in both training_image_read and testing_image_read (where x is any arbitrary integer value between 0 and 28).

training_image_read[start:end][x][y] and training_image_read[start:end][x][y] outputs the integer value at row x+1, and column y+1 in each image (between indexes start and end) in both training_image_read and testing_image_read (where x and y and arbitrary integer values between 0 and 28).

Resizing arrays:

The problem with training_image_read and testing_image_read, is that they have dimensions of (x, 28, 28) (where x can be any arbitrary integer constant), whereas I require my training array to be of dimensions (x, 784) as I am planning to make a CNN with an input layer which accepts arrays with dimensions (1, 784). I have specified my node dimension sizes above for my CNN.

In order to resize my training_image_read and testing_image_read arrays from (x, 28, 28) to (x, 784), I need to come up with an algorithm.

For resizing my arrays, one approach I can take is appending each element of the array into a one-dimensional array. This will give me an array length of 784.

Since my arrays training_image_read and testing_image_read are 3-dimensional arrays I will need 3 for loops that can iterate and fetch each integer value (between 0 and 255), which will be appended to a one-dimensional array.

My pseudocode for this approach:

```
function resizing_list(array1):
        final_output = []
        for x = 0 to len(array1)-1:
                temp1 = []
                for y = 0 to len(array1[x])-1:
                        for z = 0 to len(array1[x][y])-1:
                                temp1.append(array1[x][y][z])
                final_output.append(temp1)
        return final_output
```

My project: Handwriting recognition research project

For programming this, I will use a much more smaller sample size of **training_image_read**, because since I'm using 3 for loops, iteration can potentially take a long time to iterate if I used the full list of 60000, and so testing will be harder for me as I would have to wait a long time each time to test if my pseudocode.

```
def resizing_list(array1):
    final_output = [] #assigning an empty array to final_output, this will have a global scope so I can use to add 1-D lists in loops
    for x in range(len(array1)):# first for loop that loops between 0, and length of array1
        temp1 = []# assigning an empty array to temp1, this will have a global scope so I can use to add integer pixel values in further loop
        for y in range(len(array1[x])): # second for loop that loops between 0 and lenght of array1[x] which will be 28 in our case
            for z in range(len(array1[x][y])):# second for loop that loops between 0 and lenght of array1[x] which will be 28 in our case
                temp1.append(array1[x][y][z])# array1[x][y][z] contains the integer pixel value of each pixel in the image; there will be 784 elements in temp1
        final_output.append(temp1)# temp1 contains all integer pixel values of one image, it is added to final_output
    return final_output # returns the final list of size (len(array1), 784)
new_list = resizing_list(training_image_read[:20])# I have taken the first 20 elements of training_image_read as input for resizing_list()
```

The output of **resizing_list(training_image_read[:20])** is assigned to new_list. The new dimensions of the array are (20, 784). Therefore, my algorithm works. I can apply this algorithm to arrays **training_image_read** and **testing_image_read**.

```
In [30]: np.shape(new_training_resized)
Out[30]: (20, 784)
```

Since my **resizing_list()** function works with **training_image_read** (the first 20 elements), I will try my function on **training_image_read** and **testing_image_read**.

Also, since there are 70000 arrays that my algorithm needs to resize, I need a way of saving the new arrays into a list once they are resized, so I can use these new arrays in my neural network without having to use this program each time. Therefore, I will also add code to save my arrays (including the labels **training_label_read, testing_label_read**) in a JSON file. I am going to use a JSON file because it can store dictionaries which are easy to parse data when saving and later search (when loading). I can save all 4 of my arrays in a python dictionary and save it in a JSON file. The arrays will be easily retrieved when I need it for my neural network.

```
import json
dict_data = {"training_resized": new_training_resized,  #dict_data is a dictionary data structure with 4 items "training_resized", "testing_resized", "training_label", "testing_label" with respective lists assigned to it
             "testing_resized": new_testing_resized,
             "training_label": training_label_read.tolist(),
             "testing_label": testing_label_read.tolist()}
f = open("MNIST_resized_ready_for_nn", "w") #created a file called "MNIST_resized_ready_for_nn", this is where I will save my dictionary dict_data.
data_storage = json.dump(dict_data, f) #the dump()method from JSON is used to write in the python dictionary into the file "MNIST_resized_ready_for_nn"
f.close() #closes file "MNIST_resized_ready_for_nn5"
```

Since **training_label_read** and **testing_label_read** are numpy arrays (array of uint8 (see Spyder Variable Explorer above)), **JSON.dump()** cannot serialise it. Hence I have to use the **.tolist()** to convert it to a list, so that it can be serialised.

Test	Input data/Input code	Expected Result	Actual Result	Pass/Fail	Explanation/Justification
Loading 4 arrays of the MNIST dataset	I will code in the Python shell	Dict_data = {"training_resized": array of dimensions (60000, 784),	Dict_data = {"training_resized": array of dimensions (60000, 784),	Pass	I expected this to pass because I have worked with serialising JSON in the past and I know that if you load a JSON file

(training image array, training label array, testing image array, testing label array)	dict_data = json.load()	“testing_resized”: array of dimensions (10000, 784), “training_label”: array of dimensions (60000), “testing_label”: array of dimensions (10000) }	“testing_resized”: array of dimensions (10000, 784), “training_label”: array of dimensions (60000), “testing_label”: array of dimensions (10000) }		using json.load(), it outputs a dictionary data structure.

My code for resizing MNIST dataset:

```
import gzip
training_image_file = gzip.open("train-images-idx3-ubyte.gz", "rb")#finds and opens local file directory of training images
training_label_file = gzip.open("train-labels-idx1-ubyte.gz", "rb")#finds and opens local file directory of training labels
testing_image_file = gzip.open("t10k-images-idx3-ubyte.gz", "rb")#finds and opens local file directory of testing images
testing_label_file = gzip.open("t10k-labels-idx1-ubyte.gz", "rb")#finds and opens local file directory of testing labels
import idx2numpy
training_image_read = idx2numpy.convert_from_file(training_image_file) #reads from the training images IDX file and will output a
training_label_read = idx2numpy.convert_from_file(training_label_file) #reads from the training labels IDX file and will output a
testing_image_read = idx2numpy.convert_from_file(testing_image_file) #reads from the testing images IDX file and will output a Nu
testing_label_read = idx2numpy.convert_from_file(testing_label_file) #reads from the testing labels IDX file and will output a Nu
training_image_file.close() # closing training images file
training_label_file.close() # closing training labels file
testing_image_file.close() # closing testing image file
testing_label_file.close() # closing testing labels file
def resizing_list(array1):
    final_output = [] #assigning an empty array to final_output, this will have a global scope so I can use to add 1-D lists in l
    for x in range(len(array1)):# first for loop that loops between 0, and length of array1
        temp1 = []# assigning an empty array to temp1, this will have a global scope so I can use to add integer pixel values in
        for y in range(len(array1[x])): # second for loop that loops between 0 and lenght of array1[x] which will be 28 in our ca
            for z in range(len(array1[x][y])):# second for loop that loops between 0 and lenght of array1[x] which will be 28 in
                temp1.append(array1[x][y][z]/255)# array1[x][y][z] contains the integer pixel value of each pixel in the image; t
        final_output.append(temp1)# temp1 contains all integer pixel values of one image, it is added to final_output
    return final_output # returns the final list of size (len(array1), 784)
new_training_resized = resizing_list(training_image_read)# I have taken the first 20 elements of training_image_read as input for
new_testing_resized = resizing_list(testing_image_read)
import json
dict_data = {"training_resized": new_training_resized,  #dict_data is a dictionary data structure with 4 items "training_resized"
             "testing_resized": new_testing_resized,
             "training_label": training_label_read.tolist(),
             "testing_label": testing_label_read.tolist())
f = open("MNIST_resized_ready_for_nn", "w") #created a file called "MNIST_resized_ready_for_nn", this is where I will save my dic
data_storage = json.dump(dict_data, f) #the dump()method from JSON is used to write in the python dictionary into the file "MNIST_resized_ready_for_nn"
f.close() #closes file "MNIST_resized_ready_for_nn"
```

I have decided to normalise the MNIST dataset, because as explained in **Analysis of existing Machine Learning Programs**, normalising converts my dataset from categoric to continuous data, which is better to train on for machine learning. The way I have normalised data is dividing each pixel value in the dataset (which is an integer value from 0 to 255) by 255, which will create a continuous value.

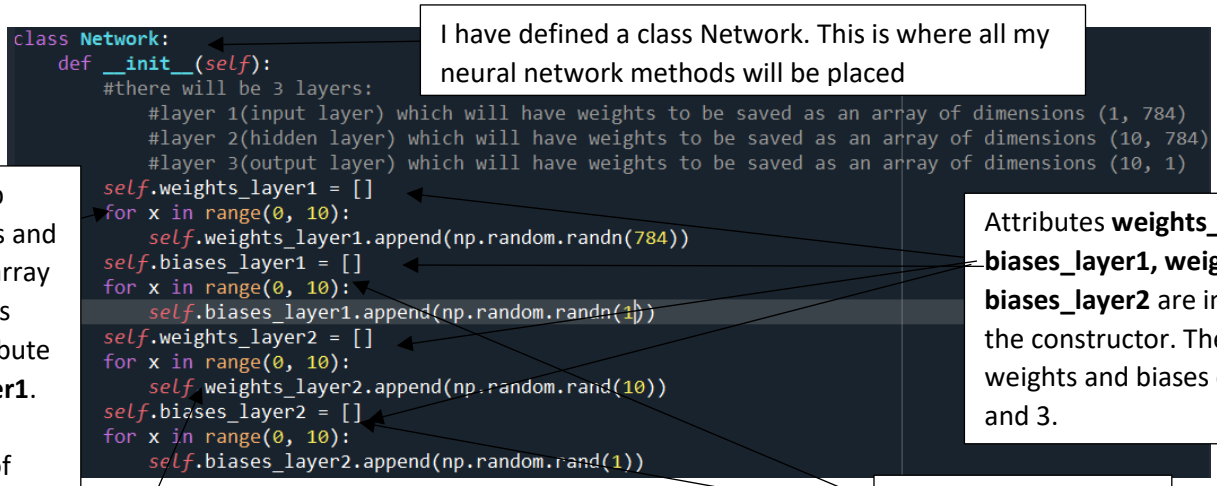
Section 3.1.2) Initializing my neural network weights and biases

use them to in my neural network.

As seen in **Analysis of Existing Machine Learning Programs**, the first thing that is needed to do is initialise random values for the neural network, which will be changed later (in iterations) when processing and doing calculations with MNIST dataset data.

I have implemented OOP (Object Oriented Programming)

My project: Handwriting recognition research project



```
class Network:
    def __init__(self):
        #there will be 3 layers:
            #layer 1(input layer) which will have weights to be saved as an array of dimensions (1, 784)
            #layer 2(hidden layer) which will have weights to be saved as an array of dimensions (10, 784)
            #layer 3(output layer) which will have weights to be saved as an array of dimensions (10, 1)
        self.weights_layer1 = []
        for x in range(0, 10):
            self.weights_layer1.append(np.random.randn(784))
        self.biases_layer1 = []
        for x in range(0, 10):
            self.biases_layer1.append(np.random.randn(1))
        self.weights_layer2 = []
        for x in range(0, 10):
            self.weights_layer2.append(np.random.rand(10))
        self.biases_layer2 = []
        for x in range(0, 10):
            self.biases_layer2.append(np.random.rand(1))
```

I have defined a class Network. This is where all my neural network methods will be placed

Iterative loop runs 10 times and appends an array of dimensions (784) to attribute **weights_layer1**. The final dimensions of **weights_layer1** will be of dimensions (10, 784)

Attributes **weights_layer1, biases_layer1, weights_layer2, biases_layer2** are initialised in the constructor. These are the weights and biases of layers 2 and 3.

The **biases_layer1** and **biases_layer2** will contain biases specifically for layer 1 and layer 2. Since each node needs to have 1 bias, both **biases_layer1** and **biases_layer2** will have dimensions (10,10) as both have 10 nodes and each node will have one bias array.

np.random.randn(size) makes a numpy array with the size input as parameter. It generates random gaussian distribution values and saves them as elements. **np.random.rand(size)** does the same but instead of random gaussian distribution values, the values being appended to the array are specifically between 0 and 1

Attribute **weight_layer2** will be the output layer on which the Softmax algorithm will be based on. I have initialised the array with random floating type values between 0 and 1

Test	Input data/ Input code	Expected Result	Actual Result	Pass/Fail	Explanation/Justification
Check the dimensions of my initial layers' array are as I want it to be.	I will write this in the Python shell: 1) nn = Network() 2) np.shape(nn. weights_layer1) 3) np.shape(nn. weights_layer2) 4) np.shape(nn.biases_layer1) 5) np.shape(nn.biases_layer2)	(10, 784) (10, 10) (10, 1) (10, 1)	(10, 784) (10, 10) (10, 1) (10, 1)	Pass	N/A

Test results:

```
In [2]: nn = Network()

In [3]: np.shape(nn.weights_layer1)
Out[3]: (10, 784)

In [4]: np.shape(nn.weights_layer2)
Out[4]: (10, 10)

In [5]: np.shape(nn.biases_layer1)
Out[5]: (10, 1)

In [6]: np.shape(nn.biases_layer2)
Out[6]: (10, 1)
```

Shape of weights of layer 1

Shape of weights of layer 2

Shape of biases of layer 1

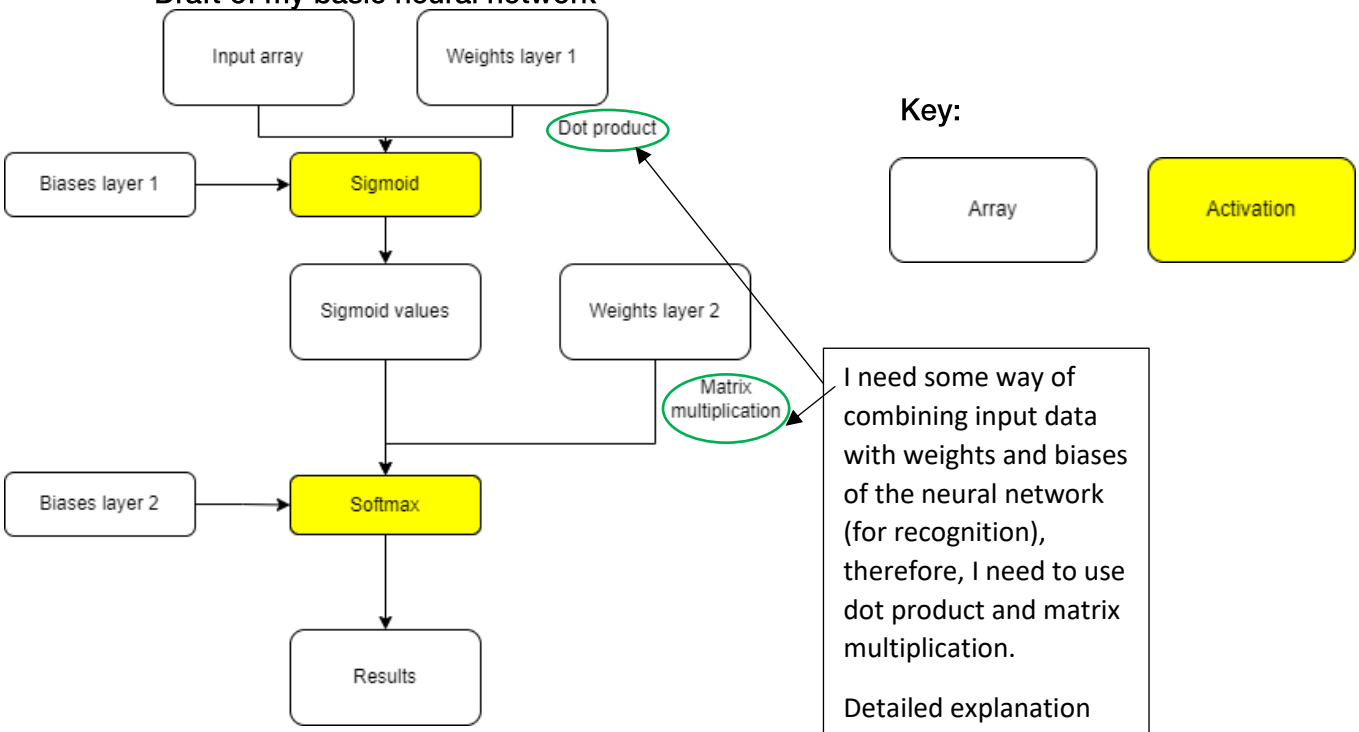
Shape of biases of layer 2

Section 3.1.3) Making Neural Network methods (excluding Backpropagation)

Backpropagation is probably the most important part of my neural network because this is where the weight and bias adjustments are done so that the neural network can learn and make accurate predictions for future iterations. Therefore, I need to make sure I build a suitable framework around backpropagation first, so I can solely focus on fine-tuning the algorithm for my backpropagation later, and rigorously test the efficacy of it to reduce the cost of the neural network (difference between the expected result of the data and the actual result of the neural network).

Therefore, I will construct a basic neural network that just feeds the input into Layer 2 (with sigmoid activation), and then Layer 3 (output) by performing SoftMax activation. The Softmax algorithm will be used for working out cost and cost differential in the backpropagation algorithm.

Draft of my basic neural network



Dot Product and Matrix multiplication:

If two arrays are of the same dimensions, then you can perform the dot product. The dimensions of the results of the dot product of two arrays is the same as the dimensions of the input arrays. In the dot product of two vectors (or arrays), each element of the first

vector of the first vector is multiplied by each element (in its equivalent position) of the second vector. The resulting vector is therefore the same dimensions.

Example:

Vector 1: $\begin{pmatrix} 23 \\ 31 \\ 45 \\ 68 \end{pmatrix}$ Vector 2: $\begin{pmatrix} 16 \\ 10 \\ 43 \\ 79 \end{pmatrix}$

Dot product of Vector 1 and Vector 2: $\begin{pmatrix} 23 * 16 \\ 31 * 10 \\ 45 * 43 \\ 68 * 79 \end{pmatrix} = \begin{pmatrix} 368 \\ 310 \\ 1935 \\ 5372 \end{pmatrix}$

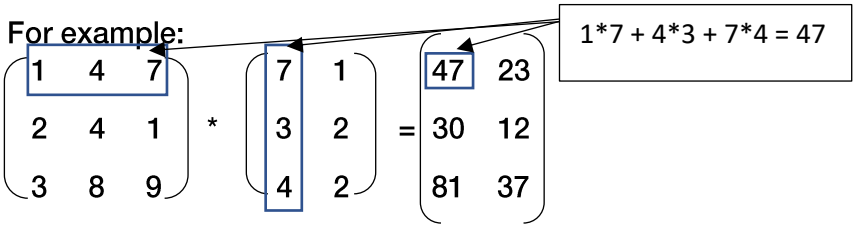
Matrix Multiplication is where two matrices (which I can model as arrays in my code) are multiplied. However, for matrix multiplication the following condition must be met:

- The number of columns in Matrix 1 must equal the number of rows in Matrix 2.

The output matrix will have the same number of rows as matrix 1 and same number of columns as matrix 2.

To find the result between two matrices, you have to multiply each element in each row in matrix 1 by each corresponding element in each column of matrix 2 and then add the results.

For example:

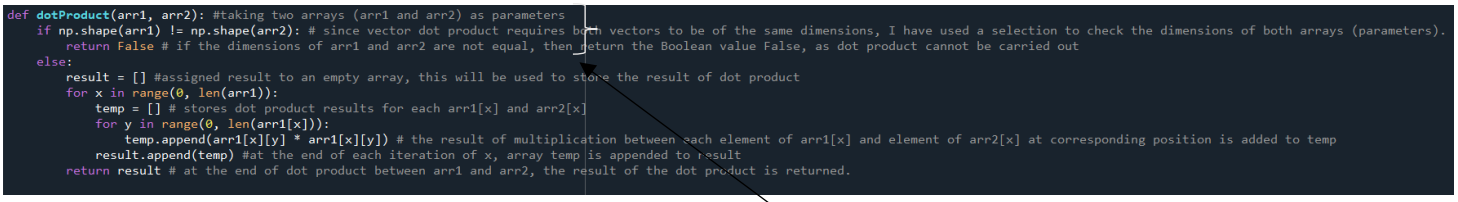


I will use Vector dot product for Layer 2, as the sizes of the input layer and my node weight arrays in Layer 2 are equal.

For Layer 3, the dimensions of my Layer 3 weights are (10, 10), and the dimension of my Layer 2 results are (10, 784), where the number of columns of Layer 3 weights is equal to the number of rows of Layer 2 results, so I can use matrix multiplication.

My project: Handwriting recognition research project

This is my source code for the vector dot product of two arrays. I have not put this function in my **Network()** class, because this is a mathematical function and isn't a structural part of my neural network.



```
def dotProduct(arr1, arr2): #taking two arrays (arr1 and arr2) as parameters
    if np.shape(arr1) != np.shape(arr2): # since vector dot product requires both vectors to be of the same dimensions, I have used a selection to check the dimensions of both arrays (parameters).
        return False # if the dimensions of arr1 and arr2 are not equal, then return the Boolean value False, as dot product cannot be carried out
    else:
        result = [] #assigned result to an empty array, this will be used to store the result of dot product
        for x in range(0, len(arr1)):
            temp = [] # stores dot product results for each arr1[x] and arr2[x]
            for y in range(0, len(arr1[x])):
                temp.append(arr1[x][y] * arr1[x][y]) # the result of multiplication between each element of arr1[x] and element of arr2[x] at corresponding position is added to temp
            result.append(temp) #at the end of each iteration of x, array temp is appended to result
        return result # at the end of dot product between arr1 and arr2, the result of the dot product is returned.
```

This is an internal validation that I am performing to make sure that both parameters of my function **arr1** and **arr2** are the same dimensions. This will prevent my function from outputting any erroneous results and will ensure the reliability of dot producting (which is crucial for my neural network).

Since, I will mainly be using **dotProduct()** for arrays of dimensions (x , 784), I have coded the iterations in this function so that it can accommodate those lists in particular.

I will test my **dotProduct()** function by generating two random arrays (using **np.random.rand()**) of a smaller sample size (1, 10) (instead of (1, 784)) and test it against my own working out of the dot product.

Test on **dotProduct()**

Input arrays:

```
In [2]: arr1 = []

In [3]: arr1.append(np.random.rand(10))

In [4]: arr2 = []

In [5]: arr2.append(np.random.rand(10))
```

arr1 = [[0.182274,

0.515038,

0.308353,

0.993834,

0.858768,

0.206277,

0.612241,

0.745856,

0.484041,

0.0576178]]

arr2 = [[0.636559,

0.427374,

0.269871,

0.807997,

0.709651,

0.278796,

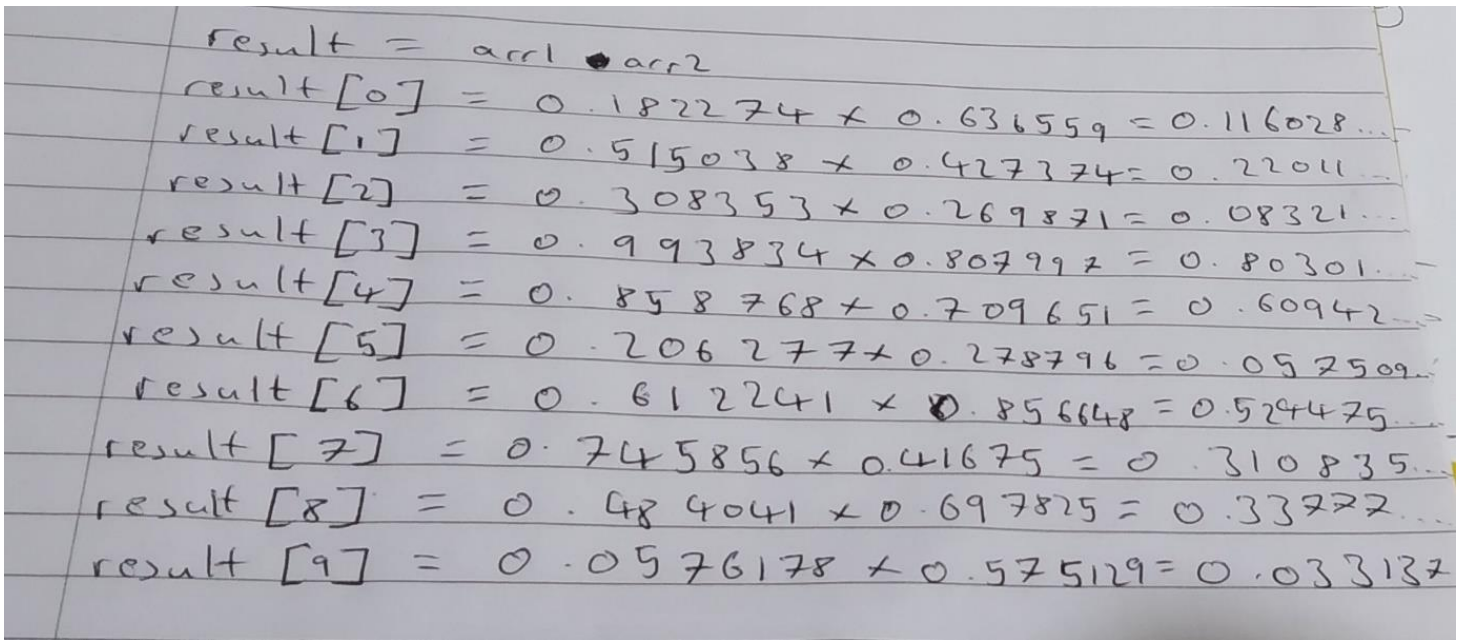
0.856648,

0.41675,

0.697825,

0.575129]]

My working out (expected result):



Code result (actual result):

My project: Handwriting recognition research project

```
In [9]: dotProduct(arr1, arr2)
Out[9]:
[[0.03322385393505538,
  0.26526410215511886,
  0.09508158251319179,
  0.9877050409748988,
  0.737482479666456,
  0.04255000372711632,
  0.37483947636025544,
  0.5563011358828134,
  0.2342953393638585,
  0.003319805406846861]]
```

Test result: Fail

Explanation/Justification: My function **dotProduct()** gives the wrong output. The problem lies in the line **temp.append(arr1[x][y] * arr1[x][y])**, where the square of **arr1[x][y]** is worked out and appended to **temp** (which is then added to **result**). This is wrong.

The appropriate code should by **temp.append(arr1[x][y] * arr2[x][y])**, where each element in **arr1** is multiplied by each corresponding element in **arr2** and the result is appended to **temp**.

Before (failed test):

```
def dotProduct(arr1, arr2): #taking two arrays (arr1 and
    if np.shape(arr1) != np.shape(arr2): # since vector d
        return False # if the dimensions of arr1 and arr2
    else:
        result = [] #assigned result to an empty array, t
        for x in range(0, len(arr1)):
            temp = [] # stores dot product results for ea
            for y in range(0, len(arr1[x])):
                temp.append(arr1[x][y] * arr1[x][y]) # th
            result.append(temp) #at the end of each itera
        return result # at the end of dot product between
```

arr1[x][y]*arr1[x][y] works out the square of **arr1[x][y]** not dot product.

After (changes made to fix failed test):

```
def dotProduct(arr1, arr2): #taking two arrays (arr1 a
    if np.shape(arr1) != np.shape(arr2): # since vecto
        return False # if the dimensions of arr1 and a
    else:
        result = [] #assigned result to an empty array
        for x in range(0, len(arr1)):
            temp = [] # stores dot product results for
            for y in range(0, len(arr1[x])):
                temp.append(arr1[x][y] * arr2[x][y]) #
            result.append(temp) #at the end of each it
        return result # at the end of dot product betw
```

arr1[x][y]*arr2[x][y] works out multiplication of **arr1[x][y]** and **arr2[x][y]**, and follows the dot product method.

Test on **dotProduct()** (attempt 2):

Input arrays:

My project: Handwriting recognition research project

```
In [2]: arr1 = []

In [3]: arr1.append(np.random.rand(10))

In [4]: arr2 = []

In [5]: arr2.append(np.random.rand(10))
```

arr1 = [[0.0894542,

0.00233087,

0.34045,

0.797264,

0.35851,

0.978308,

0.13547,

0.0936133,

0.30849,

0.746708]]

arr2 = [[0.721966,

0.0758321,

0.952789,

0.761215,

0.82817,

0.989068,

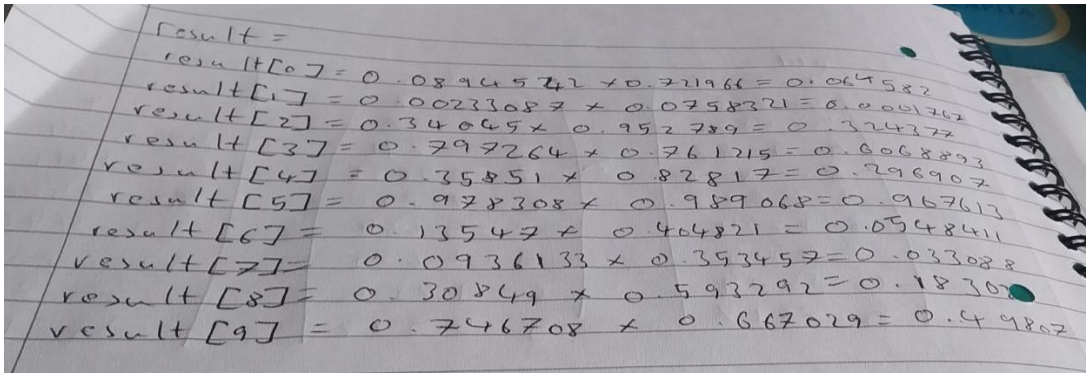
0.404821,

0.353457,

0.593292,

0.667029]]

My working out (expected result):



Code output (actual result):

```
In [6]: result = dotProduct(arr1, arr2)

In [7]: result
Out[7]:
[[0.06458291741687883,
  0.00017675465226349863,
  0.32437722029832844,
  0.6068891454478106,
  0.2969073178623609,
  0.9676132237563824,
  0.054841327181643584,
  0.03308824716560958,
  0.18302488354446217,
  0.4980757205828877]]
```

Test result: Pass

Explanation/Justification: After I made the changes to my **dotProduct()** function, the output is now correct. The code results match up with my working out, so I can now use it on a much bigger test sample size (weights of my neural network layer 1, and input image array, which are (10, 784), and (1, 784) in dimensions respectively).

I have also made the **sigmoid()** function, where I will perform sigmoid activation once I have performed the dot product of weights (layer 1) and the input image array.

Note: this is a function and not a method of class **Network()** because **sigmoid()** is a mathematical function and isn't a structural component of my neural network, hence I do not want to define it as a method of class **Network()**.

```
def sigmoid(x): #function sigmoid() takes x as a parameter where x is a floating point number
    return 1 / (1 + np.exp(-x)) #since sigmoid is 1/(1+e^(-x)), I have used np.exp() for euler's number, where np.exp() takes x as a parameter.
```

In section **Section 2.2.1**, I designed a method called feedforward(), which was a method in the Network class where both sigmoid and softmax activations took place. I am going to change that slightly and split the method into two separate methods **layer1_to_layer2()** and **layer2_to_layer3()** where I will perform dot product, and sigmoid with attribute **weights_layer1** and parameter **inputArr** in **layer1_to_layer2()**, and softmax and matrix multiplication in **layer2_to_layer3()**. I have done this to further decompose my

feedforward() so it is more modular, and I can do more precise testing with each part and fine tune each section.

Below is my method layer1_to_layer2() of class Network(), where the weights of layer 1 will be updated based on input (which is parameter inputArr). It also uses the functions sigmoid() and dotProduct().

```
    def layer1_to_layer2(self, inputArr):
        dotProduct_l1w_inpArr = [] # currently empty but will contain all the dot product arrays between self.weights_layer1 and inputArr
        for x in range(0, self.weights_layer1): # since the dimension of self.weights_layer1 is (10, 784) and the size of inputArr is (1, 784), I wil need to dot product all the 10 arrays of self.weights_layer
            dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) # every array (of length 784) of self.weights_layer1 is dot product with inputArr, and the result is assigned to dotProduct_temp
            dotProduct_l1w_inpArr.append(dotProduct_temp) # each dotProduct_temp is added to dotProduct_l1w_inpArr
```

This is where **inputArr** is dot producted with each node (array) of **self.weights_layer1**. This is saved in list **dotProduct_l1w_inpArr**

```
# adding bias to dotProduct_l1w_inpArr
n_values_with_added_biases = [] # currently empty but will contain all the dotProduct_l1w_inpArr elements with added self.biases_layer1 elements.
for x in range(0, len(dotProduct_l1w_inpArr)):
    temp = []
    for y in range(0, len(dotProduct_l1w_inpArr[x])):
        added_bias = dotProduct_l1w_inpArr[x][y] + self.biases_layer1[x][0] # since self.biases_layer1 has dimensions (10, 1) but dotProduct_l1w_inpArr has dimensions (10, 784), I need to add the bias of each each node to all 784 elements of dotProduct_l1w_inpArr
        temp.append(added_bias)
    n_values_with_added_biases.append(added_bias)
```

This is where biases (self.biases_layer1) of each node is added to the dot product **dotProduct_l1w_inpArr**. This is assigned to the list **n_values_with_added_biases**.

```
# performing sigmoid on n_values_with_added_biases
new_weights = [] # currently empty but will contail all the arrays of n_values_with_added_biases once sigmoid is applied
for x in range(0, len(n_values_with_added_biases)):
    temp = [] # empty array which will contain the values of sigmoid activation done on array of n_values_with_added_biases[x]
    for y in range(0, len(n_values_with_added_biases[x])):
        sigmoid_val = sigmoid(n_values_with_added_biases[x][y]) # sigmoid is applied to every floating point element of n_values_with_added_biases[x]
        temp.append(sigmoid_val)
    new_weights.append(temp)
self.layer1_activation = new_weights # I created a new attribute layer1_activation that stores the activation of layer 1 when inputArr, and weights_layer1 are passed through sigmoid activation
```

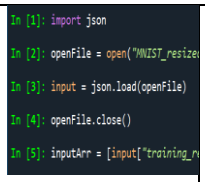
The **sigmoid()** function is applied to each element of **n_values_with_added_biases**.

This is assigned to **new_weights**. At the end of the method, the attribute **weights_layer1** is updated by replacing it with the values of **new_weights**. I have made a new attribute that saves this activation of **inputArr** and **self.weights_layer1**. It is called **layer1_activation**.

Test on layer1_to_layer2()

Test	Input code/array	Expected result	Actual result	Pass/Fail	Explanation/Justification

My project: Handwriting recognition research project

Mathematical operations are performed correctly on input in the method layer1_to_layer2()		Runs without errors and outputs weight arrays with dimensions (10, 784)	Error: 	Fail	I did not get any output for new weights_layer_1, instead I got an error because I accidentally assigned an array for the bounds of a for loop in my layer1_to_layer2() method.

Fixing my failed test:

The problem in my method layer1_to_layer2() was the line for x in range(0, self.weights_layer1): where I assigned the bounds of the iterative loop between 0 and attribute weight_layer1 which is a list, not an integer. Therefore, I get a TypeError when I invoke that method. I will change this line to for x in range(0, len(self.weights_layer1)): which will make my bounds (for iteration) between two positive integers. The purpose of the iteration was to perform dot product between weight array values for every node in layer 1 with the inputArr, and save it in a new array dotProduct_l1w_inpArr.

Before:

```
def layer1_to_layer2(self, inputArr):
    dotProduct_l1w_inpArr = [] # currently empty but will contain all the dot pro
    for x in range(0, self.weights_layer1): # since the dimension of self.weights
        dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) # every ar
        dotProduct_l1w_inpArr.append(dotProduct_temp) # each dotProduct_temp is a
    # adding bias to dotProduct_l1w_inpArr
```

The parameters of **range()** need to be integers not a list.

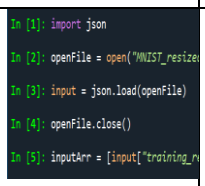
After:

My project: Handwriting recognition research project

```
def layer1_to_layer2(self, inputArr):
    dotProduct_l1w_inpArr = [] # currently empty but will contain all the dot
    for x in range(0, len(self.weights_layer1)): # since the dimension of sel
        dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) # ever
        dotProduct_l1w_inpArr.append(dotProduct_temp) # each dotProduct_temp
    # adding bias to dotProduct_l1w_inpArr
```

I have changed the parameters of **range()** so the interpreter doesn't output a **TypeError** for this specific line and iterative loop.

Test on layer1_to_layer2() (attempt 2)

Test	Input code/array	Expected result	Actual result	Pass/Fail	Explanation/Justification
Mathematical operations are performed correctly on input in the method layer1_to_layer2()		Runs without errors and outputs weight arrays with dimensions (10, 784)	Error: 	Fail	This is an error that I wasn't expecting. The TypeError means that there is at least one element in the list dotProduct_l1w_inpArr that is a Boolean value, hence why I cannot carry out an iteration (as the len()

					doesn't output an integer for Boolean values).

Fixing my failed test:

The fact that at least one of my elements in **dotProduct_l1w_inpArr** is a boolean value means that at some point in my method **layer1_to_layer2()** my inputs for **dotProduct()** must have failed the validation, which returns a boolean if the dimensions of my parameters of **dotProduct()** are not the same dimensions. To find out why my program failed the **dotProduct()** validation, I will add print statements to output the dimensions of each parameter of **dotProduct()**. Since I only used **dotProduct()** in the first iterative loop of **layer1_to_layer2()**, I will only put print statements between those lines.

```
for x in range(0, len(self.weights_layer1)): # since the dimension of s
    dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) # ev
    dotProduct_l1w_inpArr.append(dotProduct_temp) # each dotProduct_ter
```

This is the only place (so far) that I have used **dotProduct()**. So I need to check the dimensions of attribute **weights_layer1[x]** and **inputArr**.

The shape of inputArr is (1, 784), which is what I expected and wanted:

```
In [12]: np.shape(inputArr)
Out[12]: (1, 784)
```

Therefore, the problem must lie with the dimensions of attribute **weights_layer1**

This is my code after adding print statements:

```
def layer1_to_layer2(self, inputArr):
    dotProduct_l1w_inpArr = [] # currently empty but will contain all the dot product arrays between self.weights_layer1 and
    print("before dotProduct, the dimensions of self.weights_layer1: " + str(np.shape(self.weights_layer1)))
    for x in range(0, len(self.weights_layer1)): # since the dimension of self.weights_layer1 is (10, 784) and the size of in
        print("during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = {}: ".format(x) + str(np.shape(self.weights_layer1[x])) + ", " + str(np.shape(inputArr)))
        dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) # every array (of length 784) of self.weights_layer1 is dot product with inputArr, and the result is assigned to do
        print("during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = {}:".format(x) + str(dotProduct_temp))
        dotProduct_l1w_inpArr.append(dotProduct_temp) # each dotProduct_temp is added to dotProduct_l1w_inpArr
    print("final state of dotProduct_l1w_inpArr: " + str(dotProduct_l1w_inpArr))
    # adding bias to dotProduct_l1w_inpArr
    n_values_with_added_biases = [] # currently empty but will contain all the dotProduct_l1w_inpArr elements with added self.biases_layer1 elements.
```

This is just a preliminary check to make sure that the array dimensions of attribute **weights_layer1** are consistent with my predictions (which is (10, 784)).

This print statement shows the output of **dotProduct**() at each index of weights_layer1. Using this, I can see at what indexes the element is a boolean value for **dotProduct_l1w_inpArr**.

This print statement prints the dimensions of **weights_layer1** and **inputArr** during iteration (at each value of **x**). Since both, **weights_layer1** and **inputArr** are my parameters for **dotProduct**(), I can manually check where the dimensions of both parameters are inconsistent with each other and at what index of **weights_layer1**. Using this I can deduce where the problem may lie in my programming.

My project: Handwriting recognition research project

This is my output for the method layer1_to_layer2() (excluding the TypeError):

Preliminary check passed

```
In [19]: nn.layer1_to_layer2(inputArr)
before dotProduct, the dimensions of self.weights_layer1: (10, 784)
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 0: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 0:False
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 1: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 1:False
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 2: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 2:False
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 3: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 3:False
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 4: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 4:False
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 5: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 5:False
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 6: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 6:False
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 7: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 7:False
during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = 8: (784,), (1, 784)
during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = 8:False
during dotProduct, the dimensions of self.weights_layer1[x] and i
during dotProduct, the output of dotProduct(self.weights_layer1[x
final state of dotProduct_l1w_inpArr: [False, False, False, False
```

As you can see, for all the indexes of **weights_layer1**, the dimensions of **weights_layer1** and **inputArr** are inconsistent, meaning all the elements of **dotProduct_l1w_inpArr** will be Boolean False (as the validation of **dotProduct()** will fail).

One change I can do to make all the dimensions of weights_layer1 and inputArr consistent is put dotProduct([self.weights_layer1[x]], inputArr) instead of dotProduct(self.weights_layer1[x], inputArr). This will make sure that both parameters of dotProduct() will have the dimensions of (1,784).

Before:

```
for x in range(0, len(self.weights_layer1)): # since the dimension of self.w
    print("during dotProduct, the dimensions of self.weights_layer1[x] and
    dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) # every
    print("during dotProduct, the output of dotProduct(self.weights_layer1[x
    dotProduct_l1w_inpArr.append(dotProduct_temp) # each dotProduct_temp is
print("final state of dotProduct_l1w_inpArr: " + str(dotProduct_l1w_inpArr))
```

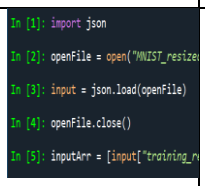
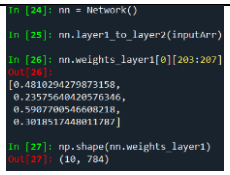
After:

```
for x in range(0, len(self.weights_layer1)): # since the dimension of sel
    print("during dotProduct, the dimensions of self.weights_layer1[x] an
    dotProduct_temp = dotProduct([self.weights_layer1[x]], inputArr) # ev
    print("during dotProduct, the output of dotProduct(self.weights_layer
    dotProduct_l1w_inpArr.append(dotProduct_temp) # each dotProduct_temp
print("final state of dotProduct_l1w_inpArr: " + str(dotProduct_l1w_inpAr
```

I have changed the **dotProduct(self.weights_layer1[x], inputArr)** to **dotProduct([self.weights_layer1[x]], inputArr)**, to fix validation issues in **dotProduct()**

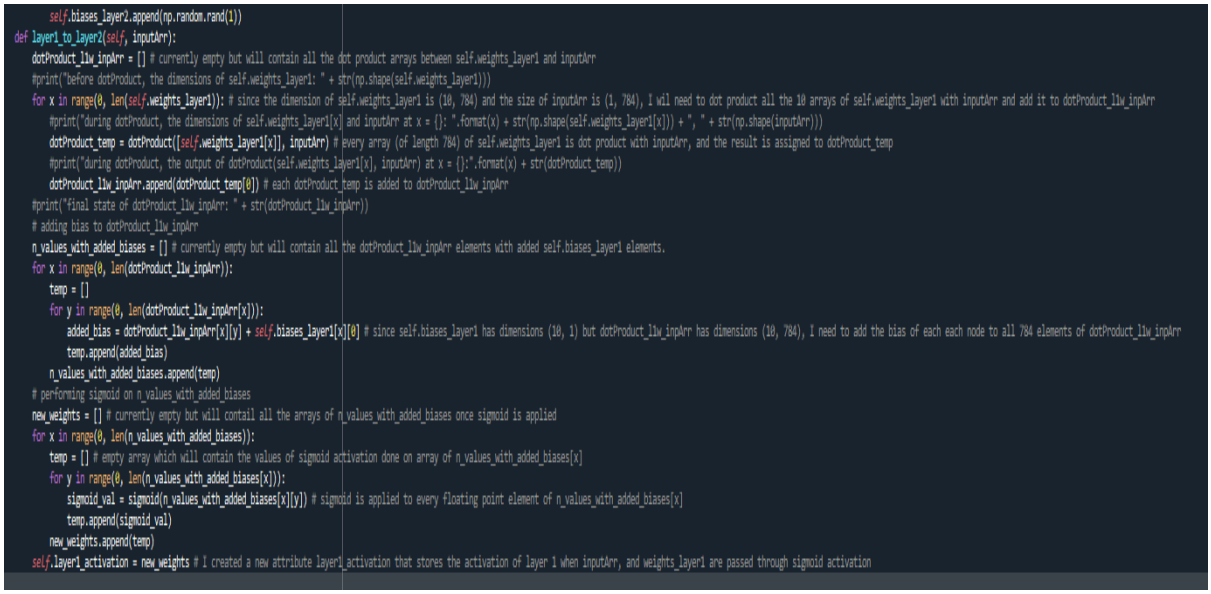
My project: Handwriting recognition research project

Test on layer1_to_layer2() (attempt 3)

Test	Input code/array	Expected result	Actual result	Pass/Fail	Explanation/Justification
Mathematical operations are performed correctly on input in the method layer1_to_layer2()		Runs without errors and outputs weight arrays with dimensions (10, 784)		Pass	There are no errors in my method layer1_to_layer2() and the new weights_layer1 are the same dimensions as initial weights of layer 1. Therefore, I can conclude layer1_to_layer2() is working as expected.

Final code for method layer1_to_layer2():

My project: Handwriting recognition research project



```
        self.biases_layer2.append(np.random.rand(1))
def layer1_to_layer2(self, inputArr):
    dotProduct_l1w_inpArr = [] # currently empty but will contain all the dot product arrays between self.weights_layer1 and inputArr
    #print("before dotProduct, the dimensions of self.weights_layer1: " + str(np.shape(self.weights_layer1)))
    for x in range(0, len(self.weights_layer1)): # since the dimension of self.weights_layer1 is (10, 784) and the size of inputArr is (1, 784), I wil need to dot product all the 10 arrays of self.weights_layer1 with inputArr and add it to dotProduct_l1w_inpArr
        #print("during dotProduct, the dimensions of self.weights_layer1[x] and inputArr at x = {}: ".format(x) + str(np.shape(self.weights_layer1[x])) + ", " + str(np.shape(inputArr)))
        dotProduct_temp = dotProduct([self.weights_layer1[x]], inputArr) # every array (of length 784) of self.weights_layer1 is dot product with inputArr, and the result is assigned to dotProduct_temp
        #print("during dotProduct, the output of dotProduct(self.weights_layer1[x], inputArr) at x = {}:".format(x) + str(dotProduct_temp))
        dotProduct_l1w_inpArr.append(dotProduct_temp[0]) # each dotProduct_temp is added to dotProduct_l1w_inpArr
    #print("final state of dotProduct_l1w_inpArr: " + str(dotProduct_l1w_inpArr))
    # adding bias to dotProduct_l1w_inpArr
    n_values_with_added_biases = [] # currently empty but will contain all the dotProduct_l1w_inpArr elements with added self.biases_layer1 elements.
    for x in range(0, len(dotProduct_l1w_inpArr)):
        temp = []
        for y in range(0, len(dotProduct_l1w_inpArr[x])):
            added_bias = dotProduct_l1w_inpArr[x][y] + self.biases_layer1[x][0] # since self.biases_layer1 has dimensions (10, 1) but dotProduct_l1w_inpArr has dimensions (10, 784), I need to add the bias of each each node to all 784 elements of dotProduct_l1w_inpArr
            temp.append(added_bias)
        n_values_with_added_biases.append(temp)
    # performing sigmoid on n_values_with_added_biases
    new_weights = [] # currently empty but will contail all the arrays of n_values_with_added_biases once sigmoid is applied
    for x in range(0, len(n_values_with_added_biases)):
        temp = [] # empty array which will contain the values of sigmoid activation done on array of n_values_with_added_biases[x]
        for y in range(0, len(n_values_with_added_biases[x])):
            sigmoid_val = sigmoid(n_values_with_added_biases[x][y]) # sigmoid is applied to every floating point element of n_values_with_added_biases[x]
            temp.append(sigmoid_val)
        new_weights.append(temp)
    self.layer1_activation = new_weights # I created a new attribute layer1_activation that stores the activation of layer 1 when inputArr, and weights_layer1 are passed through sigmoid activation
```

Next, I need to work on applying the Softmax activation for matrix (result of matrix multiplication between attributes **weights_layer1** and **weights_layer2**). For this, I will need to make an algorithm to do the matrix multiplication first.

I will write pseudocode for matrix multiplication:

For context: array = [[1,2,3,4], [3,2,3,4], [1,2,4,5]] in matrix form:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 3 & 4 \\ 1 & 2 & 4 & 5 \end{pmatrix}$$

Function matrixMultiplication(mat1, mat2): //parameters mat1 and mat2 represent mat1 multiplied by mat2 (in that order)

row_mat1 = len(mat1)

row_mat2 = len(mat2)

column_mat1 = len(mat1[0])

column_mat2 = len(mat2[0])

If column_mat1 != row_mat2 then //this is validation, if the number of columns of mat1 doesn't equal to the number of rows of mat2, then matrix multiplication cannot be done, so output a Boolean False.

return False

else:

final_dim = (row_mat1, column_mat2) // the final dimensions of matrix multiplication always has the rows of mat1 and columns of mat2

final_list = []

My project: Handwriting recognition research project

for x = 0 to row_mat1-1: //making an empty matrix of dimensions final_dim

temp_row = []

for y = 0 to column_mat2-1:

temp_row.append(0) // columns created with initialised value of 0

endfor

final_list.append(temp_row)

endfor

for x = 0 to row_mat1-1: //since I need multiply all the elements in a row of mat1 by the elements in every column of mat2, I will need a nested for loop with one iterating in the number of rows of mat1 and one iterating in the number of columns of mat2

for y = 0 to column_mat2-1:

total = 0 //for each mat1 row that is multiplied by mat2 column will need a way to work out the running sum (cumulative sum).

for z = 0 to column_mat1-1 // z keeps a track of what mat1[row mat1][column mat 1] is being multiplied with mat2[row mat2][column mat2]

total = total + mat1[x][z] * mat2[z][y] // running total

final_list[x][y] = total // the dimensions of final_list has already been made, so I just substitute matrix multiplication sums in its right position in the final_list

return final_list // result of matrix multiplication is returned.

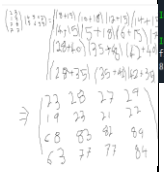
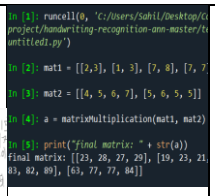
My code for matrix multiplication (function **matrixMultiplication()**):

```
def matrixMultiplication(mat1, mat2):
    row_mat1 = len(mat1) # row of mat1
    row_mat2 = len(mat2) # row of mat2
    column_mat1 = len(mat1[0]) # column of mat1
    column_mat2 = len(mat2[0]) # column of mat2
    if column_mat1 != row_mat2: # validation of matrices mat1 and mat2
        return False # if number of columns of mat1 isnt the same as number of rows of mat2, matrix multiplication cannot be done
    else:
        final_dim = (row_mat1, column_mat2) # the final dimensions of matrix multiplication
        final_list = []
        for x in range(0, row_mat1):
            row = [] # each row of the final matrix (there are row_mat1 number of rows)
            for y in range(0, column_mat2):
                row.append(0) # each row will be initialised with 0s
            final_list.append(row) # rows with 0s is added to final_list
        #final list is now made with the dimensions of final_dim
        for x in range(0, row_mat1): # I will need a nested loop because I am multiplying each row of mat1 by each corresponding column of mat2, so one loop will iterate in row_mat1, and one will iterate in column_mat2
            for y in range(0, column_mat2):
                total = 0 # cumulative sum for every value of the final result
                for z in range(0, column_mat1): # we need a loop to keep track of column of mat1 so we can use it to multiply each mat1[row][column (which is z)] with mat2
                    total += mat1[x][z] * mat2[z][y] # running total
                final_list[x][y] = total # when one row of mat1 is fully multiplied with column of mat2 and summed up, the value is substituted in the final_list[row mat1][column mat2].
        return final_list # results of matrix multiplication is returned
```

Testing of matrixMultiplication()

Test	Input data/ Input code	Expected result	Actual result	Pass/ Fail	Explanation

My project: Handwriting recognition research project

Carry out the following matrix multiplication: Matrix 1: $\begin{pmatrix} 2 & 3 \\ 1 & 3 \\ 7 & 8 \\ 7 & 7 \end{pmatrix}$ Matrix 2: $\begin{pmatrix} 4 & 5 & 6 & 7 \\ 5 & 6 & 5 & 5 \end{pmatrix}$	mat1 = [[2,3], [1, 3], [7, 8], [7, 7]] mat2 = [[4, 5, 6, 7], [5, 6, 5, 5]] a = matrixMultiplication(mat1, mat2) print("final matrix" + str(a))	My working out: 		Pass	I expected my input data to pass the validation of **matrixMultiplication()** because the column length of mat1 is equal to the row length of mat2. I was unsure whether my program will solve the matrix multiplication correctly because there are so many iterative loops and element retrieval of **mat1** and **mat2** for calculations, that I hadn't really tested prior to this, therefore I was expecting at least **Syntax** or **Type** errors. Thankfully, the function has passed my first test.
I will enter two parameters that should fail the initial validation of the function. Matrix 1: $\begin{pmatrix} 1 & 2 \\ 3 & 7 \\ 5 & 9 \end{pmatrix}$ Matrix 2:	mat1 = [[1, 2], [3, 7], [5, 9]] mat2 = [[1], [3], [4]] a = matrixMultiplication(mat1, mat2) print("final matrix: " + str(a))	"final matrix: False"	"final matrix: False"	Pass	This testing proves to me that my validation of parameters **mat1** and **mat2** is working and will prevent any errors further down in the function **matrixMultiplication()**.

1 3 5					
I will test my function matrixMultiplication() by adding attributes layer1_activation and weights_layer2 as parameters and checking if there are any errors for generating an output. This test is crucial for my neural network because my neural network will use this output for matrixMultiplication() for Softmax activation.	nn = Network() result = matrixMultiplication (nn.weights_layer1, nn.weights_layer2) print(np.shape(result))	(10, 784)	()	Fail	I have realised that this my mistake (for user input). For my function matrixMultiplication() to generate a result array of dimensions (10,784), I must multiply attribute weights_layer2 by layer1_activation because row length of weights_layer2 is 10, and the column length of layer1_activation is 784. The final dimensions of any matrix multiplication is (row length of matrix 1, column length of matrix 2).

Fixing my failed test (part 3):

Since this is a user input error (my mistake), the solution of this failed test is swapping the parameters values of mat1 and mat2 (in function) so it is weights_layer2 multiplied by layer1_activation not layer1_activation multiplied by weights_layer2. This will give me my desired output:

```
In [9]: result = matrixMultiplication(  nn.weights_layer2,nn.layer1_activation)

In [10]: print(np.shape(result))
(10, 784)
```

My project: Handwriting recognition research project

Now that I have a matrix multiplication function (matrixMultiplication()), I can work on feeding in values into the Softmax activation, and give an output for layer 3 (the output layer where all the node values add up to 1).

Before starting to work with Softmax, I have two main problems I will need to solve, for which I will need validation:

- The final array of activation of layer 3 must be of dimension (10, 1) otherwise, the functions like the cost function, and backpropation will not work.
- All the node values of layer 3 activation must add up to 1, because each node represents a probability, and the probability distribution of the neural network isn't mathematically valid if all the nodes of layer 3 activation won't add up to 1.

First of all, I need to make a function for the Softmax algorithm. The formula of Softmax is:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Since I am inputting an array of dimension (10, 784) and expecting an array of dimension (10, 1) in return, I need z (from the above formula) to be equal to each node in layer 2 (i.e. each element array in Softmax input). If I use each node as a value of z, I can work out what probability each node in layer 2 makes of all the nodes in layer 2. This will give me an output (array) which will contain elements that will add up to 1.

However, the problem I have is that since z is an array of dimension (784), I can't raise it as a power of Euler's number, because for e^z, z needs to be a number not an array(mathematically). Since the output I need is just a probability (which (in the most basic terms) is what proportion of a list an element is), I can make z equal to the sum of all 784 elements of each individual node, which means that the denominator will be the sum of all 784 elements of each 10 nodes (raised to Euler's number). I think this will give me a feasible output for Softmax algorithm.

This is my code representation of the approach I will take:

```
def softmax(arr1): # arr1 represents matrix multiplication between attributes weights_layer2 and layer1_activation
    size_arr1 = (10, 784)
    # to output an array of size (10, 1) we need to divide each sum of nodes (as power of e) by the total sum of nodes (as power of e)
    list_sum_of_each_node = []
    for x in range(0, len(arr1)):
        temp = 0
        for y in range(0, len(arr1[x])):
            temp += arr1[x][y]
        list_sum_of_each_node.append(np.exp(temp)) # since each node has 784 elements, we can use the sum() method to add up each element
    total = 0 # total sum of all nodes and its elements(as power of e)
    for x in range(0, len(list_sum_of_each_node)):
        total += list_sum_of_each_node[x]
    result = []
    for x in range(0, len(arr1)):
        result.append(list_sum_of_each_node[x]/total) # division of sum of each node (as power of e) by total sum of nodes (as power of e)
    return result # output of softmax
```

My project: Handwriting recognition research project

Test on **softmax()**:

Test	Input data/input code	Expected output	Actual output	Pass/Fail	Explanation
Test if the Softmax algorithm works properly on weights of layer 2. I will test this by seeing if the elements of output of **softmax()** add up to 1.	nn = Network() result = softmax(nn.weights_layer2) total = 0 for x in range(0, len(result)): total += result[x] print(total)	1	0.9999999999999996	Pass	While (mathematically speaking) 0.99999999 is not 1 but its close enough that I can deduce, that the sum is indeed very close to 1. This proves to me that my **softmax()** works and I can implement it in my neural network.

My **layer2_to_layer3()** method:

Below is **my layer2_to_layer3(),** which will apply Softmax to weights of layer 2 and existing weights of layer 3. The outputs of **softmax()** will be saved as a new a **layer2_activation** which will be used in backpropagation later.

```
def layer2_to_layer3(self):
    matrixMult_l2w_l1 = matrixMultiplication(self.weights_layer2, self.weights_layer1) # matrix multiplication of weights of weights_layer2 (layer 3) and weights of weights_layer1 (layer 2)
    n_values_with_added_biases = [] # currently empty but will contain all the matrixMult_l2w_l1 elements with added self.biases_layer2 elements.
    for x in range(0, len(matrixMult_l2w_l1)):
        temp = []
        for y in range(0, len(matrixMult_l2w_l1[x])):
            added_bias = matrixMult_l2w_l1[x][y] + self.biases_layer2[x][0] # since self.biases_layer2 has dimensions (10, 1) but matrixMult_l2w_l1 has dimensions (10, 784), I need to add the bias of each each node to all 784 elements of matrixMult_l2w_l1[x]
            temp.append(added_bias)
        n_values_with_added_biases.append(temp)
    final_probabilities = softmax(n_values_with_added_biases) # Softmax activation applied here, the output will be a list of probabilities (dimension (10))
    for x in range(len(final_probabilities)):
        final_probabilities[x] = [final_probabilities[x]] # this will convert the dimensions of final_probabilities from (10) to (10, 1)
    self.layer2_activation = final_probabilities # I created a new attribute layer2_activation that stores the activation of layer 2 when weights_layer1, and weights_layer2 are passed through softmax activat
```

This should be **self.layer1_activation**, not **self.weights_layer1**, in the later iterations of the code this is corrected.

This is where matrix multiplication is done between attributes **weights_layer2** and **layer1_activation**

This is where bias (attribute **bias_layer2**) is added to output of **matrixMultiplication()** **matrixMult_l2w_l1**

This is where the Softmax activation (**softmax()**) is applied to **n_values_with_added_biases**

Since the dimensions output array of **softmax()** is (10) and not (10, 1), I have added an iteration that changes each element of **final_probabilities** from a floating point to a 1-Dimensional list containing that float number

Test on layer2_to_layer3()

Test	Input code/array	Expected result	Actual result	Pass/Fail	Explanation/Justification
Mathematical operations are performed correctly on input in the method layer2_to_layer3() given that layer2_to_layer3() is already done	In Python Shell openFile = open("MNIST_resized_ready_for_nn") import json input = json.load(openFile) inputArr = [input["training_resized"][0]] nn = Network() nn.layer1_to_layer2(inputArr) nn.layer2_to_layer3() np.shape(nn.weights_layer2)	Runs without errors and outputs probability arrays with dimensions (10, 1)	(10, 1)	Pass	I anticipated that this method would pass my test because there isn't any new functionality or any complex operation that I have added. This function mainly involves calling functions such as matrixMultiplication() and softmax(). The part where I applied biases to the matrixMult_l2w_l1 uses similar commands and logic as layer1_to_layer2(), so I also didn't expect any errors there as well.

Section 3.1.4) Making the Backpropagation algorithm

As I researched in Section 1.4.1(Machine Learning Research), Backpropagation is used to minimize the error/cost (which is the difference between the target output and the neural network output) of the neural network. Minimizing the error of the neural network means that the neural network output is closer to the target output. This will mean that the neural network is more accurate in predicting outcome than the previous epoch, and is hence more useful.

For my neural network, the cost/error function (I will call it cost function) will be $Cost = (prediction - target)^2$. The closer the value of Cost is to 0, the more accurate the

neural network is. 0 is the turning point of the Cost equation meaning that $\frac{d(Cost)}{dx} = 0$ (where x = prediction-target). The derivative of the Cost equation ($\frac{d(Cost)}{dx}$) gives the gradient of the parabola at a given x, and also gives us how much we need to increase/decrease the Cost equation by to get to the minimum point (turning point y = 0). We can use this information to backpropagate in the neural network and increment/decrement the initial weights of the layers.

We will use the $\frac{d(Cost)}{dx}$ to work out $\frac{d(Cost)}{d(weights\_layer3)}, \frac{d(Cost)}{d(weights\_layer2)}, \frac{d(Cost)}{d(biases\_layer3)}, \frac{d(Cost)}{d(biases\_layer2)}$ which tell the neural network how much to increment/decrement each weight of layer 2, weight of layer 3, bias of layer 2, bias of layer 3 (respectively).

To work out $\frac{d(Cost)}{d(weights\_layer3)}, \frac{d(Cost)}{d(weights\_layer2)}, \frac{d(Cost)}{d(biases\_layer3)}, \frac{d(Cost)}{d(biases\_layer2)}$ we need to use chain rule, because working out the Cost in relation to weights and biases isn't straightforward at all, and will require knowledge of partial derivative fractions.

The closest layer we have to the output layer is the output layer (layer 3), so we need to work out $\frac{d(Cost)}{d(weights\_layer3)}$ first to find out how much layer 3 is to be nudged by, for the next iteration.

$\frac{d(Cost)}{d(weights\_layer3)} = \frac{d(Cost)}{dx} * \frac{dx}{d(layer3\_activation)} * \frac{d(layer3\_activation)}{d(weights\_layer3)}$ is the equation for working out $\frac{d(Cost)}{d(weights\_layer3)}$. As you can see dx, and d(layer3_activation) cancel each other out, which will simplify down to $\frac{d(Cost)}{d(weights\_layer3)}$.

For working out $\frac{d(Cost)}{d(weights\_layer2)}$, it will be a bit more complicated because to access weights of layer 2, you will have to access weights of layer 3 first and traverse back using that. So, the equation for $\frac{d(Cost)}{d(weights\_layer2)}$ will be:

$$\frac{d(Cost)}{d(weights\_layer2)} = \frac{d(Cost)}{dx} * \frac{dx}{d(layer3\_activation)} * \frac{d(layer3\_activation)}{d(layer2\_activation)} * \frac{d(layer2\_activation)}{d(weights\_layer2)}$$

This is my interpretation of the backpropagation algorithm, however as I have explained in the **Design section (section 2.2.1)** that I will be referring to and using parts of a neural network in Simple MNIST NN from scratch (numpy, no TF/Keras) | Kaggle for backpropagation. This is because, I think I lack the knowledge and skill set to come up with my own algorithm for such a complicated area of machine learning, and I am also limited by the time scale of this project.

This is their function for backpropagation (**backward_prop()**):

For context, A1 is their activation where they have used the ReLU function for weights of layer 2 (dot producted with input array), A2 is the activation where they have performed Softmax with layer 2 weights (multiplied with A1).

My project: Handwriting recognition research project

Also, one_hot_Y is referring to the array that contains the desired output for each input array. They have converted Y, which is an integer (corresponding output for input array so it is between 0-9) to an array of length 10, where the element at the desired outcome index is 1 and the rest of the elements are 0. This is important because it allows you to work out the cost function for each of the 10 nodes of output, as each of the 10 nodes will have an output and a corresponding desired output.

X and Y refer to their input array, and its corresponding target output.

m is the size of MNIST dataset (X) that the author has used.

```
def backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y):
    one_hot_Y = one_hot(Y)
    dZ2 = A2 - one_hot_Y
    dW2 = 1 / m * dZ2.dot(A1.T)
    db2 = 1 / m * np.sum(dZ2)
    dZ1 = W2.T.dot(dZ2) * ReLU_deriv(Z1)
    dW1 = 1 / m * dZ1.dot(X.T)
    db1 = 1 / m * np.sum(dZ1)
    return dW1, db1, dW2, db2
```

This is the differential of the cost function in relation to x so $\frac{d(Cost)}{dx}$ will equal to (prediction (**A2**) – target (**one_hot_Y**)). For context, they have used the same Cost equation as me

dB1 and **dB2** both have similar mathematical methods of working out. They are just assigned to averages (mean) of **dZ1** and **dZ2** (which part of error incrementation/decrement of **W1** and **W2**), meaning biases will be nudged by an average error interval for both weights (**W1**, and **W2**).

The **dW2** (this will contain the values that are to be incremented/decremented later to weights of layer 3) chain rule partial fractions are similar to my interpretations (see above). The **dZ2** is the same as my $\frac{d(Cost)}{dx}$ and they are multiplying that by **A1**. In my interpretation of $\frac{d(Cost)}{d(weights\_layer3)}$ I used $\frac{dx}{d(layer3\_activation)} * \frac{d(layer3\_activation)}{d(weights\_layer3)}$ in my calculation as the **layer3_activation** (Softmax) would cancel out to give **weight_layer3** as a denominator (which is what is desired). However, in the code adaptation above, they have just multiplied $\frac{d(Cost)}{dx}$ by **A1**, which is the ReLU activation with layer 2 weights, and not included Softmax (which is their layer 3 activation).

The **dW1** (this will contain the values that are to be incremented/decremented later to weights of layer 2) chain rule partial fractions are also similar to my interpretations (see above). In my interpretation of $\frac{d(Cost)}{d(weights\_layer2)}$ I used $\frac{d(layer3\_activation)}{d(layer2\_activation)} * \frac{d(layer2\_activation)}{d(weights\_layer2)}$ in my calculation as the **layer3_activation** (Softmax) and **layer2_activation** (sigmoid) would cancel out to give **weight_layer2** as a denominator (which is what is desired). Similarly, in the code adaptation above they have used **ReLU_deriv()** which is their function containing the derivative of ReLU (their layer 2 activation), so they have also used $\frac{d(layer2\_activation)}{d(weights\_layer2)}$.

My project: Handwriting recognition research project

This is my code interpretation of backpropagation:

First of all I need to make a cost and cost derivative method as this is crucial for partial derivatives for backpropagation of layer 2 and layer 3 weights and biases. I have therefore made a method that outputs both cost and cost derivative so it can be used inside backpropagation but also later on for testing the error of the neural network.

This is my **cost_and_cost_deriv()** method:

```
def cost_and_cost_deriv(self, inputLabel): # this method will be used inside backpropagation for chain rule but also for testing if the error is minimised or not
    final_cost_function_arr = [] # this will contain result of (prediction - target)^2, I intend to make its dimensions (10, 1)
    final_cost_deriv_function_arr = [] # this will contain result of derivitive of cost 2*(prediction - target), I intend to make its dimensions (10, 1)
    for x in range(0, 10): # since there are 10 output nodes, so I have a made a loop that iterates 10 times
        final_cost_function_arr.append([(self.weights_layer2[x][0] - inputLabel[x])**2]) # cost of node appended to final_cost_function_arr
        final_cost_deriv_function_arr.append([(self.weights_layer2[x][0] - inputLabel[x])]) # cost derivative of node appended to final_cost_deriv_function_arr
        return final_cost_function_arr, final_cost_deriv_function_arr # I have returned both things, so the output of cost_and_cost_deriv() will be a list of size 2 containing 2 (10, 1) arrays
```

I also need a method for the derivative of sigmoid function. This will be represented as the partial derivative $\frac{d(layer2\_activation)}{d(weights\_layer2)}$ for working out $\frac{d(Cost)}{d(weights\_layer2)}$. The derivative of sigmoid is $\frac{d\,(sigmoid)}{dx} = \frac{1}{(1+e^{-x})} * (1 - \frac{1}{1+e^{-x}})$. My function **sigmoid_prime()** is below:

```
def sigmoid_prime(x): #function sigmoid_prime() takes x as a parameter where x is a floating point number
    return sigmoid(x)*(1-sigmoid(x)) #since sigmoid is 1/(1+e^(-x)), I have used np.exp() for euler's number, where np.exp() takes x as a parameter.
```

I also needed to make a function for transposing matrices (this is where the rows and columns of a matrix are swapped). There are instances in the code above from Simple MNIST NN from scratch (numpy, no TF/Keras) | Kaggle where certain matrix multiplication cannot be performed because the columns of matrix 1 isn't equal to rows of matrix 2, however transposing is required to make the matrix multiplication work. For example, if there is a matrix of dimension (10, 1) (where 10 is the number of rows and 1 is the number of columns) and a matrix of dimension (35,1), we can transpose the second matrix so its dimensions become (1, 35) and so the number of columns of matrix 1 (1) is equal to the number of rows in matrix 2 (1).

Here is my **transpose()** function:

```
def transpose(arr1): # for backpropagation, certain matrix multiplication calculations wouldn= not work unless you transpose them, so this is my implementation
    final_list = [] # currently empty but will contain results of transposing
    for x in range(0, len(arr1[0])): # I will iterate in the number of column first so I can locate element at every arr1 row at a given column and put them in a new list
        temp = [] # will contain elements from every row for every given column(x)
        for y in range(0, len(arr1)):
            temp.append(arr1[y][x]) # elements from every row for every given row(x) is appended to temp
        final_list.append(temp) # the new row is added to final_list()
    return final_list # the results are outputted
```

This is my main **backpropagation()** method (in which I have implemented the alg (not syntax) of Simple MNIST NN from scratch (numpy, no TF/Keras) | Kaggle):

These are the values that will be incremented/decremented onto **weights_layer1, weights_layer2, biases_layer1, biases_layer2**. This should minimise cost intervals for future iterations.

My project: Handwriting recognition research project

```
def backpropagation(self, inputArr, inputLabel): # the only parameters I need are inputArr (dimensions (1,784)) and inputLabel (contains desired output) of dimensions (10)
    dCost_dx = cost_and_cost_deriv(inputLabel)[1] # this represents my partial derivative d(cost)/d(x)
    dWeight2 = 1/1 * matrixMultiplication(transpose(self.layer1_activation) ,dCost_dx) # matrix multiplication will result in dimensions (784, 1)
    dBias2 = 1/1 * np.sum(dCost_dx) # bias error margin uses the sum of dCost_dx
    sigmoid_prime_layer1 = [] # currently empty but will contail all the arrays of weights_layer1 once sigmoid is applied, dimension will be (10, 784)
    for x in range(0, len(self.weights_layer1)):
        temp = [] # empty array which will contain the values of sigmoid activation done on array of self.weights_layer1[x]
        for y in range(0, len(self.weights_layer1[x])):
            sigmoid_val = sigmoid(self.dotP_inputArr_weights1_with_bias[x][y]) # sigmoid is applied to every floating point element of self.weights_layer1[x]
            temp.append(sigmoid_val)
        sigmoid_prime_layer1.append(temp)
    #dotProduct(self.weights_layer1, dWeight2)
    transposed_weights_layer2 = transpose(self.weights_layer2) # self.weights_layer2 need to be transposed so dot product with dWeight2 of dimension (784, 1) can be done.
    matrixMult_W2_dCost_dx = matrixMultiplication(transposed_weights_layer2, dCost_dx) # this is a part of dActivation_layer1, we had to transpose weights_layer2 so that matrix multiplication is possible
    dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1 # we use the error interval from layer 3 so that we can work out the error interval at layer 1. The dimension of the array is (10,1)
    dWeight1 = 1/1 * matrixMultiplication(dActivation_layer1, transpose(inputArr)) # this is the error interval that the initial weights of layer 2 should be nudged by.
    dBias1 = 1/1 * np.sum(dActivation_layer1) # bias error margin uses the sum of dActivation_layer1
    return dWeight1, dBias1, dWeight2, dBias2
```

To get an error interval for layer 2 weights, you need to backpropagate from layer 3, and use its error interval so that the total neural network error margin can be minimised for future iterations.

dCost_dx has dimensions (10, 1) but attribute **weights_layer1** has dimensions (10, 784), the column length of **weights_layer1** (784) isn't equal to row length of **dCost_dx** (10). If I transpose **weights_layer1,** the dimensions of **weights_layer1** become (784, 10) where the column length of **weights_layer1** is equal to row length of **dCost_dx** (10).

Test 1 on **backpropagation()** method:

Test	Input data/input code	Expected result	Actual result	Pass/Fail	Explanation
I want to check if there are any obvious **Syntax, Type, or other errors**	nn = Network() randomInput = np.random.rand(1, 784) nn.layer1_to_layer2(randomInput) nn.layer2_to_layer3() a = nn.backpropagation(randomInput, [1,0,0,0,0,0,0,0,0,0]) print(np.shape(a[0]), np.shape(a[1]), np.shape(a[2]), np.shape(a[3]))	4 arrays containing **dWeight1, dBias1, dWeight2, dBias2**	d(x) NameError: name 'cost_and_cost_deriv' is not defined	Fail	I forgot to put "self." infront of **cost_and_cost_deriv()** because it is a method of the Network() not a function.

My project: Handwriting recognition research project

Fixing my test:

Before:

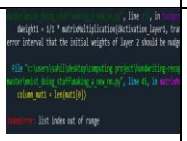
```
dCost_dx = cost_and_cost_deriv(inputLabel)[1] #
dWeight2 = 1/1 * matrixMultiplication(transpose(
```

After:

```
dCost_dx = self.cost_and_cost_deriv(inputLabel)[1] #
dWeight2 = 1/1 * matrixMultiplication(transpose(self
```

I added a "self." to call the method **cost_and_cost_deriv().**

Test 1 on backpropagation() method (attempt 2):

Test	Input data/input code	Expected result	Actual result	Pass/ Fail	Explanation/Justification
I want to check if there are any obvious Syntax, Type, or other errors	nn = Network() randomInput = np.random.rand(1, 784) nn.layer1_to_layer2(randomInput) nn.layer2_to_layer3() a = nn.backpropagation(randomInput, [1,0,0,0,0,0,0,0,0,0]) print(np.shape(a[0]), np.shape(a[1]), np.shape(a[2]), np.shape(a[3]))	4 arrays containing dWeight1, dBias1, dWeight2, dBias2		Fail	This error has occurred in the matrixMultiplication() function for the line dWeight1 = 1/1 * matrixMultiplication(dActivation_layer1, transpose(inputArr)). One potential solution I can use to solve this problem is to take out the 1/1 out of the line as I don't think Python accepts multiplication of scalar values with arrays (especially multidimensional arrays).

Fixing my test:

To fix the Index error, I will remove 1/1 from the line dWeight1 = 1/1 * matrixMultiplication(dActivation_layer1, transpose(inputArr)) and retry the test.

Before:

```
dWeight2 = 1/1 * matrixMultiplication(transpose(self.layer1_activation) ,dCost_dx)
dBias2 = 1/1 * np.sum(dCost_dx) # bias error margin uses the sum of dCost_dx
```

```
dWeight1 = 1/1 * matrixMultiplication(dActivation_layer1, transpose(inputArr))
dBias1 = 1/1 * np.sum(dActivation_layer1) # bias error margin uses the sum of d
```

All 4 of these lines contain **1/1**. I included **1/1** because the code for backpropagation from Simple MNIST NN from scratch (numpy, no TF/Keras) | Kaggle included **1/m** for each important partial fraction, where **m** represents the number of input arrays in the dataset. I don't really need to use **1/m** currently as I am still testing if my backpropagation works. So I will delete the references to **1/1**

My project: Handwriting recognition research project

After:

```
dWeight2 = matrixMultiplication(transpose(self.layer
dBias2 = np.sum(dCost_dx) # bias error margin uses t
sigmoid_prime_layer1 = [] # currently empty but will
```

```
dWeight1 = matrixMultiplication(dActiva
dBias1 = np.sum(dActivation_layer1) # b
```

I have removed all references to **1/1**

Test 1 on **backpropagation()** method (attempt 3):

Test	Input data/input code	Expected result	Actual result	Pass/ Fail	Explanation/Justification

I want to check if there are any obvious **Syntax, Type, or other errors**	nn = Network()
	
	randomInput = np.random.rand(1, 784)
	
	nn.layer1_to_layer2(randomInput)
	
	nn.layer2_to_layer3()
	
	a = nn.backpropagation(randomInput, [1,0,0,0,0,0,0,0,0,0])
	
	print(np.shape(a[0]), np.shape(a[1]), np.shape(a[2]), np.shape(a[3]))
		4 arrays containing **dWeight1, dBias1, dWeight2, dBias2**	`dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1 # We use the error interval from layer 3 so that we can work out the error interval at layer 1. The dimension of the array is (10,1)`
			
			`TypeError: can't multiply sequence by non-int of type 'list'`
				Fail	This time there is a **Type** error because I multiplied two lists **matrixMult_W2_dCost_dx** and **sigmoid_prime_layer1** in the line **dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1**. I think the problem is that Python doesn't allow multiplication of two lists using the * operator. However, if you multiply two NumPy arrays using the * operator, Python will output a matrix multiplication with the two arrays. I must use my **matrixMultiplication()** to do this operation.

Fixing my test:

My project: Handwriting recognition research project

To fix the **Type error**, I will remove **matrixMult_W2_dCost_dx * sigmoid_prime_layer1**, from the line **dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1** and use my **matrixMultiplication()** instead.

Before:

```
dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1 # we
```

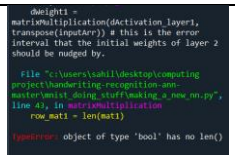
I used the * operator because I wanted to emulate the author of the code from Simple MNIST NN from scratch (numpy, no TF/Keras) | Kaggle but I forgot that the author is explicitly using NumPy arrays so they can use * with their arrays. Not all arrays of my program are NumPy arrays, so this same operation may not work for me.

After:

```
matrixMult_W2_dCost_dx = matrixMultiplication(transposed_weights_layer2, dCost_dx) # this
dActivation_layer1 = matrixMultiplication(matrixMult_W2_dCost_dx, sigmoid_prime_layer1) #
dWeight1 = matrixMultiplication(dActivation_layer1, transpose(inputArr)) # this is the err
```

I replaced **matrixMult_W2_dCost_dx * sigmoid_prime_layer1** with **matrixMultiplication(matrixMult_W2_dCost_dx, sigmoid_prime_layer1)**.

Test 1 on **backpropagation()** method (attempt 4):

Test	Input data/input code	Expected result	Actual result	Pass/Fail	Explanation/Justification
I want to check if there are any obvious **Syntax, Type, or other**	nn = Network() randomInput = np.random.rand(1, 784) nn.layer1_to_layer2(randomInput) nn.layer2_to_layer3() a = nn.backpropagation(randomInput, [1,0,0,0,0,0,0,0,0,0]) print(np.shape(a[0]), np.shape(a[1]),	4 arrays containing **dWeight1, dBias1, dWeight2, dBias2**		Fail	The line or error is **dWeight1 = matrixMultiplication(dActivation_layer1, transpose(inputArr))**. The fact that the object is Boolean (as outputted by the Python shell error) means that the matrix multiplication involving **dActivation_layer1** did not meet the validation check. I will have to

errors	np.shape(a[2]), np.shape(a[3]))				somehow solve this issue.

Fixing my test:

I do not know the dimension sizes of **matrixMult_W2_dCost_dx** and **sigmoid_prime_layer1** to deduce what operation to perform on them (matrix multiplication, dot product, or use * with NumPy arrays (converting) as a last resort (as it covers a wide range of array multiplication operations)) to assign to **dActivation_layer1**.

I will put print statements in the method **backpropagation()** to deduce the dimensions of the arrays.

```
transposed_weights_layer2 = transpose(self.weights_layer2) # self.weights_layer2 need to b
matrixMult_W2_dCost_dx = matrixMultiplication(transposed_weights_layer2, dCost_dx) # this
dActivation_layer1 = matrixMultiplication(matrixMult_W2_dCost_dx, sigmoid_prime_layer1) #
print("matrixMult_W2_dCost_dx shape: " + str(np.shape(matrixMult_W2_dCost_dx)))
print("sigmoid_prime_layer1 shape: " + str(np.shape(sigmoid_prime_layer1)))
print("dActivation_layer1 shape: " + str(np.shape(dActivation_layer1)))
print("inputArr shape: " + str(np.shape(inputArr)))
```

Python shell output:

```
matrixMult_W2_dCost_dx shape: (10, 1)
sigmoid_prime_layer1 shape: (10, 784)
dActivation_layer1 shape: ()
inputArr shape: (1, 784)
```

Since **matrixMult_W2_dCost_dx** has a dimension (10, 1) and **sigmoid_prime_layer1** has a dimension (10, 784) you cannot do matrix multiplication on it as the column length of matrix **matrixMult_W2_dCost_dx** isn't equal to the row length of **sigmoid_prime_layer1**.

Since I can't use my matrix multiplication function or dot product, I will use NumPy arrays. I can convert **matrixMult_W2_dCost_dx** and **sigmoid_prime_layer1** to NumPy arrays by using the **np.array()** method and use either the * operator or if that fails then **np.dot()** method which can do various types of matrix operations with 2 arrays (matrices).

I will use the * operation after converting my arrays to NumPy arrays.

My code implementation

Before:

```
transposed_weights_layer2 = transpose(self.weights_layer2) # self.weights_layer2 need to
matrixMult_W2_dCost_dx = matrixMultiplication(transposed_weights_layer2, dCost_dx) # thi
dActivation_layer1 = matrixMultiplication(matrixMult_W2_dCost_dx, sigmoid_prime_layer1)
```

After:

```
matrixMult_W2_dCost_dx = np.array(matrixMult_W2_dCost_dx)
sigmoid_prime_layer1 = np.array(sigmoid_prime_layer1)
dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1 #
```

I can use the * operation as I am dealing with NumPy arrays now.

I have implemented the **np.array()** method to convert **matrixMult_W2_dCost_dx** and **sigmoid_prime_layer1** to NumPy arrays.

My project: Handwriting recognition research project

Test 1 on **backpropagation()** method (attempt 5):

Test	Input data/input code	Expected result	Actual result	Pass/Fail	Explanation/Justification
I want to check if there are any obvious **Syntax, Type, or other errors**	nn = Network() randomInput = np.random.rand(1, 784) nn.layer1_to_layer2(randomInput) nn.layer2_to_layer3() a = nn.backpropagation(randomInput, [1,0,0,0,0,0,0,0,0,0]) print(np.shape(a[0]), np.shape(a[1]), np.shape(a[2]), np.shape(a[3]))	4 arrays containing **dWeight1, dBias1, dWeight2, dBias2**	I get 2 arrays and 2 floating point values	Pass	Even though, I have no errors in my code, I know that my **backpropagation()** method is wrong because the 4 arrays I got as my output should be the same size as my weights of layer 1 and layer 2 and biases of layer 1 and layer 2, but they aren't.

As you can read above in the **Test 1 on backpropagation() method (attempt 5)** table, even though I have no errors I know that I need to change my whole neural network formatting just so I can accommodate this backpropagation algorithm from Simple MNIST NN from scratch (numpy, no TF/Keras) | Kaggle .

This means that I must readjust my array dimensions and make changes to my methods and functions such as **layer1_to_layer2()**, and **layer2_and_layer3()** so that it gives out activations, weights, biases, etc, in a format (array dimensions) that can be used for the backpropagation algorithm that was made in Simple MNIST NN from scratch (numpy, no TF/Keras) | Kaggle.

First thing I must do is compare the dimensions of my network arrays with the Kaggle neural network arrays

Since Kaggle allows users to edit code I can use it to check the sizes of all the arrays used in their backpropagation algorithm.

My project: Handwriting recognition research project

```python
def backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y):
    one_hot_Y = one_hot(Y)
    dZ2 = A2 - one_hot_Y
    dW2 = 1 / m * dZ2.dot(A1.T)
    db2 = 1 / m * np.sum(dZ2)
    dZ1 = W2.T.dot(dZ2) * ReLU_deriv(Z1)
    dW1 = 1 / m * dZ1 @ X.T
    print("shape of dW1: " + str(np.shape(dW1)))
    db1 = 1 / m * np.sum(dZ1)
    print("shape of A2: " + str(np.shape(A2)))
    print("shape of dZ2: " + str(np.shape(dZ2)))
    print("shape of dW2: " + str(np.shape(dW2)))
    print("shape of db2: " + str(np.shape(db2)))
    print("shape of dZ1: " + str(np.shape(dZ1)))
    print("shape of W2: " + str(np.shape(W2)))
    print("shape of Z1: " + str(np.shape(Z1)))
    print("shape of dW1: " + str(np.shape(dW1)))
    print("shape of db1: " + str(np.shape(db1)))

    return dW1, db1, dW2, db2
```

These are the print statements that I typed in to print the shape of **dW1**, **A2** (softmax activation), **dZ2**, **dW2**, **db2**, **dZ1**, **W2** (weights of layer 3), **Z1** (weights of layer 1 dot producted with input layer + bias), **dW1**, **db1**

The results (Python shell) are below:

```
0.10143902439024391
shape of dW1: (10, 784)
shape of A2: (10, 41000)
shape of dZ2: (10, 41000)
shape of dW2: (10, 10)
shape of db2: ()
shape of dZ1: (10, 41000)
shape of W2: (10, 10)
shape of Z1: (10, 41000)
shape of dW1: (10, 784)
shape of db1: ()
Iteration:  1
```

41 000 is the number of input MNIST images for the neural network. This isn't much importance to me because the author of this code has put all imaes in batch for neural network, whereas I will do it one at a time, so my image count parameter would not be more than 1.

These are the dimensions of 2 of the output of **backpropagation()** which are the same dimensions as **weights_layer1** and **weights_layer2**

I think their bias derivatives are erroneous, however, I know from their code that this is meant to represent a sum of **dZ1** and **dZ2** with dimensions (10, 1) or (10) each.

My project: Handwriting recognition research project

My array dimensions (code):

```
        sigmoid_prime_layer1 = np.array(sigmoid_prime_layer1)
        dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1 # we use the error interval
        dWeight1 = matrixMultiplication(dActivation_layer1, transpose(inputArr)) # this is the error
        dBias1 = 1/784 * np.sum(dActivation_layer1, axis = 1) # bias error margin uses the sum of dAct
        print("dCost_dx shape: " + str(np.shape(dCost_dx)))
        print("dWeight2 shape: " + str(np.shape(dWeight2)))
        print("sigmoid_prime_layer1 shape: " + str(np.shape(sigmoid_prime_layer1)))
        print("matrixMult_W2_dCost_dx shape: " + str(np.shape(matrixMult_W2_dCost_dx)))
        print("dCost_dx shape: " + str(np.shape(dCost_dx)))
        print("dActivation_layer1 shape: " + str(np.shape(dActivation_layer1)))
        print("dWeight1 shape: " + str(np.shape(dWeight1)))
        print("dBias1 shape: " + str(np.shape(dBias1)))
        return dWeight1, dWeight2, dBias1, dBias2
```

Python shell results:

```
dCost_dx shape: (10, 1)
dWeight2 shape: (784, 1)
dBias2 shape: (10, 1)
matrixMult_W2_dCost_dx shape: (10, 1)
self.layer1_activation shape: (10, 784)
self.layer2_activation shape: (10, 1)
dActivation_layer1 shape: (10, 784)
dWeight1 shape: (10, 1)
dBias1 shape: (10,)
```

Since we are drawing parallels between both coding algorithms, **dCost_dx** is the same **dZ2** for the Kaggle program. Also, **self.layer1_activation** and **self.layer2_activation** are like **A1** and **A2**. **dWeight2, dWeight1, dBias2, dBias1** represent the output of my backpropagation() method, similar to dW1, dW2, db1, db2.

One of the first things I need to change is **self.layer1_activation** and **self.layer2 activation**. In the Kaggle program, A1 contains the sum of all 784 elements of each dot product node between layer 1 weights and input array (which is then passed into sigmoid). I must do the same in order to get an array for **self.layer1_activation** of dimensions (10). The output of **self.layer1_activation** will also affect the activation of layer 3 (**self.layer2 activation**) so I need to change that as well

Both **dWeight2** and **dWeight1** which are supposed to be the incrementing/decrementing factor for neural network weights have the wrong dimensions (i.e., not the same dimensions as **weights_layer2** and **weights_layer1**). This is a warning that my backpropagation calculations are definitely wrong

My project: Handwriting recognition research project

```
def layer1_to_layer2(self, inputArr):
    dotProduct_l1w_inpArr = [] # currently empty but will contain all the dot product arrays sums
    #print("before dotProduct, the dimensions of self.weights_layer1: " + str(np.shape(self.weigh
    for x in range(0, len(self.weights_layer1)): # since the dimension of self.weights_layer1 is
        dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) # every array (of length
        dotProduct_l1w_inpArr.append(sum(dotProduct_temp)) # each sum of all 784 elements of dot
    #print(np.shape(dotProduct_l1w_inpArr))
    # adding bias to dotProduct_l1w_inpArr
    n_values_with_added_biases = [] # currently empty but will contain all
    for x in range(0, len(dotProduct_l1w_inpArr)):
        added_bias = dotProduct_l1w_inpArr[x] + self.biases_layer1[x][0] #
        n_values_with_added_biases.append(added_bias)
    self.dotP_inputArr_weights1_with_bias = n_values_with_added_biases
    # performing sigmoid on n_values_with_added_biases
    new_weights = [] # currently empty but will contail all the arrays of n
    for x in range(0, len(n_values_with_added_biases)):
        temp = [] # empty array which will contain the values of sigmoid ac
        sigmoid_val = sigmoid(n_values_with_added_biases[x]) # sigmoid is a
        temp.append(sigmoid_val)
        new_weights.append(temp)
    self.layer1_activation = np.resize(new_weights, 10) # I created a new attribute layer1_activ
```

The first change that I have done is append the sum of all 784 elements of **dotProduct_temp** instead of the 784 elements themselves. This will output an array **dotProduct_l1w_inpArr** of dimension (10)

I need **dotP_inputArr_weights1_with_bias** in **backpropagation()** as it represents **z1** from the Kaggle program /which is needed for the activation prime function for partial derivative).

I also removed a nested loop here as **n_values_with_added_biases** doesn't have dimensions (10, 784) so only one loop is required.

I have removed the nested for loop which was previously used to add bias to 784 elements of each of the 10 nodes. Now, since **dotProduct_l1w_inpArr** has only 10 floating point numbers, you can add each bias to its corresponding node directly

I used **np.resize()** method in the end so that **layer1_activation** is strictly of dimension (10).

Changing layer2_to_layer3() method:

```
def layer2_to_layer3(self):
    matrixMult_l2w_l1 = matrixMultiplication(self.weights_layer2, self.layer1_
    #return matrixMult_l2w_l1
    matrixMult_l2w_l1_sum = []
    for x in range(0, len(matrixMult_l2w_l1)):
        matrixMult_l2w_l1_sum.append(sum(matrixMult_l2w_l1[x])) # because I want a sum of all 784 elements in ea
    n_values_with_added_biases = [] # currently empty but will contain all the matrixMult_l2w_l1_sum elements wi
    for x in range(0, len(matrixMult_l2w_l1_sum)):
        added_bias = matrixMult_l2w_l1_sum[x] + self.biases_layer2[x][0] # I need to add the bias of each each n
        n_values_with_added_biases.append(added_bias)
    #print(n_values_with_added_biases)
    final_probabilities = softmax(n_values_with_added_biases) # Softmax activation applied here, the output will
    #print("final prob " + str(sum(final_probabilities)))
    self.layer2_activation = final_probabilities # I created a new attribute layer2_activation that stores the a
```

As with **layer2_to_layer3()** method, I have removed the nested for loop which was previously used to add bias to 784 elements of each of the 10 nodes. So now biases for each node can directly be added to its corresponding **matroxMult_l2w_l1** node.

I have removed the for loop that assigned each element of **final_probabilities** a list containing the element (to add extra dimension to the **final_probabilities**). Since we need **self.layer2_activation** dimension to be (10), we don't need any loops.

Since **matroxMult_l2w_l1** is of dimension (10, 784), I will find the sum of each node (containing a list of 784 elements) and append it to a new list **matrixMult_l2w_l1_sum** which will have the dimensions (10, 1)

My project: Handwriting recognition research project

Changing softmax() method:

```
def softmax(x): # the parameter x will be an arra
    exp_total = 0 # this will contain the sum of
    for index in range(0, len(x)): # iteration in
        exp_total += round(np.exp(x[index]), 6) # I will increment (x[index]) to exp_total, raised
    exp_final_prob = [] # this will store the final probabilities of each node in relation with eve
    for index in range(0, len(x)): # iterates for every index between x
        element = round(np.exp(x[index])/exp_total, 6) # this is where the softmax formula is appli
        exp_final_prob.append(element)
    return exp_final_prob # final list of probabilities
```

I have used the round() to round everything to 6 significant figures to prevent overflow errors and divisibility by zero errors. I don't think this is necessary, but this is only precautionary.

I have changed the equation from e ^(sum of each node)/ sum(e^each node) to e^(each node)/sum(e^each node).

Since we don't have a parameter of x = an array of (10, 784) dimension elements, we do not nee to work out a sum for each node. Therefore I have removed the nested for loop for that.

Changing cost_and_cost_deriv() method:

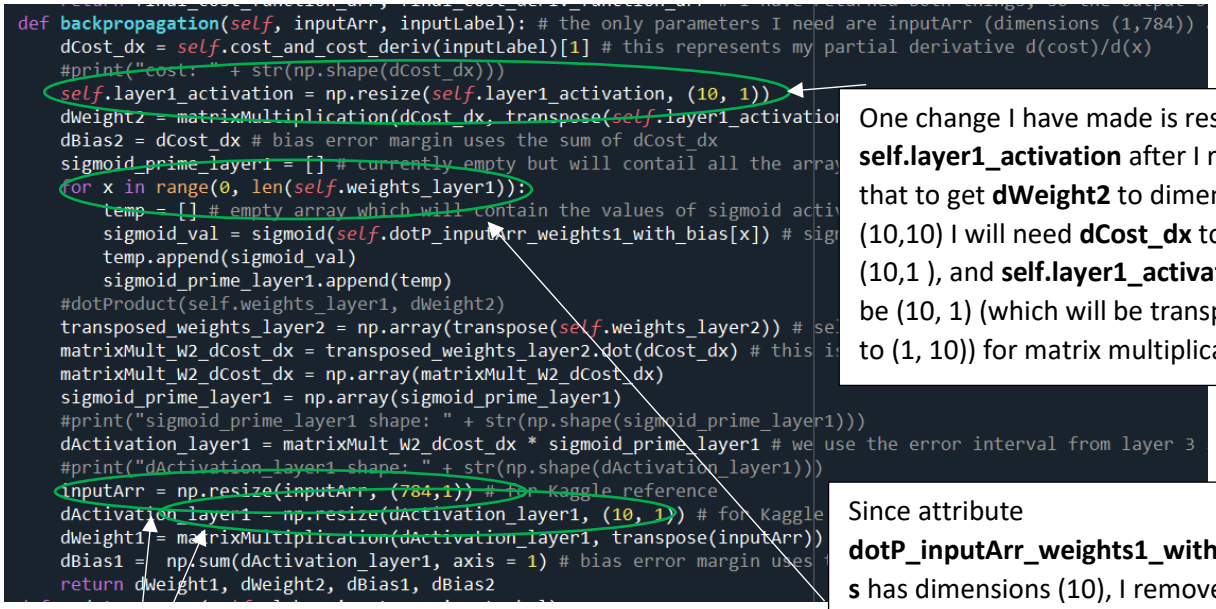
```
    self.layer2_activation = final_probabilities # I created a new attribute layer2_activation th
def cost_and_cost_deriv(self, inputLabel): # this method will be used inside backpropagation for
    final_cost_function_arr = [] # this will contain result of (prediction - target)^2, I intend
    final_cost_deriv_function_arr = [] # this will contain result of derivitive of cost 2*(predic
    for x in range(0, 10): # since there are 10 output nodes, so I have a made a loop that iterat
        final_cost_function_arr.append([(self.layer2_activation[x] - inputLabel[x])**2]) # cost c
        final_cost_deriv_function_arr.append([(self.layer2_activation[x] - inputLabel[x])]) # cos
    return final_cost_function_arr, final_cost_deriv_function_arr # I have returned both things,
```

Since **self.layer2_activation** is of dimension (10) not (10, 1), the only change I needed to make is change

self.layer2_activation[x][0] to **self.layer2_activation[x]** on both lines.

Changing backpropagation() method:

My project: Handwriting recognition research project



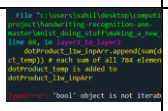
```
def backpropagation(self, inputArr, inputLabel): # the only parameters I need are inputArr (dimensions (1,784))
    dCost_dx = self.cost_and_cost_deriv(inputLabel)[1] # this represents my partial derivative d(cost)/d(x)
    #print("cost: " + str(np.shape(dCost_dx)))
    self.layer1_activation = np.resize(self.layer1_activation, (10, 1))
    dWeight2 = matrixMultiplication(dCost_dx, transpose(self.layer1_activation
    dBias2 = dCost_dx # bias error margin uses the sum of dCost_dx
    sigmoid_prime_layer1 = [] # currently empty but will contail all the array
    for x in range(0, len(self.weights_layer1)):
        temp = [] # empty array which will contain the values of sigmoid activ
        sigmoid_val = sigmoid(self.dotP_inputArr_weights1_with_bias[x]) # sigm
        temp.append(sigmoid_val)
        sigmoid_prime_layer1.append(temp)
    #dotProduct(self.weights_layer1, dWeight2)
    transposed_weights_layer2 = np.array(transpose(self.weights_layer2)) # se
    matrixMult_W2_dCost_dx = transposed_weights_layer2.dot(dCost_dx) # this i
    matrixMult_W2_dCost_dx = np.array(matrixMult_W2_dCost_dx)
    sigmoid_prime_layer1 = np.array(sigmoid_prime_layer1)
    #print("sigmoid_prime_layer1 shape: " + str(np.shape(sigmoid_prime_layer1)))
    dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1 # we use the error interval from layer 3
    #print("dActivation_layer1 shape: " + str(np.shape(dActivation_layer1)))
    inputArr = np.resize(inputArr, (784,1)) # for Kaggle reference
    dActivation_layer1 = np.resize(dActivation_layer1, (10, 1)) # for Kaggle
    dWeight1 = matrixMultiplication(dActivation_layer1, transpose(inputArr))
    dBias1 = np.sum(dActivation_layer1, axis = 1) # bias error margin uses
    return dWeight1, dWeight2, dBias1, dBias2
```

One change I have made is resizing **self.layer1_activation** after I realised that to get **dWeight2** to dimensions (10,10) I will need **dCost_dx** to be (10,1), and **self.layer1_activation** to be (10, 1) (which will be transposed to (1, 10)) for matrix multiplication

Since attribute **dotP_inputArr_weights1_with_bias** has dimensions (10), I removed the nested for loop which was previously to perform sigmoid prime on every one of the 10*784 elements.

For dWeight1 array to be (10,784), I will need to do matrix multiplication with **dActivatation_layer1**, and **inputArr**. I need the column length of **dActivation_layer1** to equal row length of **inputArr**, and the row length of **dActivation_layer1** to equal to 10, and column length of **inputArr** to equal 784. Therefore, I need to resize (using np.resize()) **dActivation_layer1** to (10, 1) and **inputArr** to (784, 1)

Test 2 on backpropagation() method:

Test	Input data/input code	Expected result	Actual result	Pass/ Fail	Explanation/Justification
I want check whether or not the outputs of backpropagation() have the correct dimensions	nn = Network() randomInput = np.random.rand(1, 784) nn.layer1_to_layer2(randomInput) nn.layer2_to_layer3() a = nn.backpropagati	4 arrays containing dWeight1, dBias1, dWeight2, dBias2 where dWeight1 has dimensions (10, 784), dBias1 has (10, 1), dWeight2		Fail	In the line dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) the dot product can't be carried out as the dimensions of self.weights_layer1[x] and inputArr are not equal. I think inputArr is (1, 784) but

	on(randomInput, [1,0,0,0,0,0,0,0,0,0]) print(np.shape(a[0]), np.shape(a[1]), np.shape(a[2]), np.shape(a[3]))	has (10, 10), and dBias2 has (10, 1)			self.weights_layer1[x] is just (784).

Fixing my test:

Before:

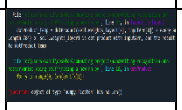
```
for x in range(0, len(self.weights_layer1)): # since the dimension
    dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr)
    dotProduct_l1w_inpArr.append(sum(dotProduct_temp)) # each sum o
#print(np.shape(dotProduct_l1w_inpArr))
```

After:

Now both parameters are of size (784).

```
for x in range(0, len(self.weights_layer1)): # since the dimension of self.w
    dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr[0]) # ever
    dotProduct_l1w_inpArr.append(sum(dotProduct_temp)) # each sum of all 784
#print(np.shape(dotProduct_l1w_inpArr))
```

Test 2 on backpropagation() method (attempt 2):

Test	Input data/input code	Expected result	Actual result	Pass/ Fail	Explanation/Justification
I want check whether or not the outputs of backpropagation() have the correct dimensions	nn = Network() randomInput = np.random.rand(1, 784) nn.layer1_to_layer2(randomInput) nn.layer2_to_layer3() a = nn.backpropagation(randomInput, [1,0,0,0,0,0,0,0,0,0]) print(np.shape(a[0]), np.shape(a[1]), np.shape(a[2]), np.shape(a[3]))	4 arrays containing dWeight1, dBias1, dWeight2, dBias2 where dWeight1 has dimensions (10, 784), dBias1 has (10, 1), dWeight2 has (10, 10), and dBias2 has (10, 1)		Fail	The Type error has occurred in the line dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr[0] (this is the line I changed for my failed test above). This is where both arrays being passed into dotProduct() are a 1 dimensional array of length 784. I think my function dotProduct() only accepts arrays that are multi-dimensional (e.g. 2 dimensional arrays).

Fixing my test:

First of all, I need to check if my suspicion is right. To replicate the same error, I will pass in two 1 dimensional NumPy arrays of length 784 (using the method np.random.rand()):

My project: Handwriting recognition research project

```
In [8]: a = np.random.rand(784)

In [9]: b = np.random.rand(784)

In [10]: dotProduct(a, b)
```

With the above code, I managed to replicate the same Type error that I got for my neural network:

```
Traceback (most recent call last):

  File "C:\Users\Sahil\AppData\Local\Temp\ipykernel_5232\733115617.py", line 1, in <cell line: 1>
    dotProduct(a, b)

  File "c:\users\sahil\desktop\computing project\handwriting-recognition-ann-master\mnist_doing_stuff\making_a_new_nn.py", line 18, in dotProduct
    for y in range(0, len(arr1[x])):

TypeError: object of type 'numpy.float64' has no len()
```

Before:

```
def dotProduct(arr1, arr2): #taking two arrays (arr1 and arr2) as parameters
    if np.shape(arr1) != np.shape(arr2): # since vector dot product requires both vectors to be of the same dimension
        return False # if the dimensions of arr1 and arr2 are not equal, then return the Boolean value False, as dot
    else:
        result = [] #assigned result to an empty array, this will be used to store the result of dot product
        for x in range(0, len(arr1)):
            temp = [] # stores dot product results for each arr1[x] and arr2[x]
            for y in range(0, len(arr1[x])):
                temp.append(arr1[x][y] * arr2[x][y]) # the result of multiplication between each element of arr1[x] a
            result.append(temp) #at the end of each iteration of x, array temp is appended to result
        return result # at the end of dot product between arr1 and arr2, the result of the dot product is returned.
```

The problem lies in this line, because, since the **x** element of arr1 will be an integer, or float and not an array, **len()** method will not give an output, hence a **Type error** is output

After:

```
def dotProduct(arr1, arr2): #taking two arrays (arr1 and arr2) as parameters
    if np.shape(arr1) != np.shape(arr2): # since vector dot product requires both vectors to be of the same dimensions, I have used a selection to check the dimensions of both arrays (parameters).
        return False # if the dimensions of arr1 and arr2 are not equal, then return the Boolean value False, as dot product cannot be carried out
    else:
        if len(np.shape(arr1)) == 1: # if there are no columns then I have to remove one of the iterative loops
            result = [] #assigned result to an empty array, this will be used to store the result of dot product
            for x in range(0, len(arr1)):
                temp = arr1[x] * arr2[x] # the result of multiplication between each element of arr1 and element of arr2 at corresponding position is added to temp
                result.append(temp)
            return result
        result = [] #assigned result to an empty array, this will be used to store the result of dot product
        for x in range(0, len(arr1)):
            temp = [] # stores dot product results for each arr1[x] and arr2[x]
            for y in range(0, len(arr1[x])):
                temp.append(arr1[x][y] * arr2[x][y]) # the result of multiplication between each element of arr1[x] and element of arr2[x] at corresponding position is added to temp
            result.append(temp) #at the end of each iteration of x, array temp is appended to result
        return result # at the end of dot product between arr1 and arr2, the result of the dot product is returned.
```

The approach I took to solve the problem regarding 1 dimensional arrays, was making an **if** statement (selection) where if **np.shape(arr1)** (which outputs the dimensions of **arr1**) is equal to 1 (i.e. is 1 dimensional) then each element of arr1 and its corresponding element in arr2 is multiplied, (the results of which are appended to an array **result**) in a single for loop. The resulting array of elements is then output.

Test 2 on backpropagation() method (attempt 3):

Test	Input data/input code	Expected result	Actual result	Pass/ Fail	Explanation/Justification
I want check whether or not the outputs of backpropagation() have the correct dimensions	nn = Network() randomInput = np.random.rand(1, 784) nn.layer1_to_layer2(randomInput) nn.layer_2_to_layer3() a = nn.backpropagation([1,0,0,0,0,0,0,0,0,0])	4 arrays containing dWeight1, dBias1, dWeight2, dBias2 where dWeight1 has dimensions (10, 784), dBias1 has (10, 1), dWeight2 has (10, 10), and dBias2 has (10, 1)	(10, 784) [dWeight 1] (10, 10)[dWeight2] (10,)[dBias1] (10, 1)[dBias2]	Pass	Even though dBias1 is not output as a 2 dimensional array (10, 1), I can fix this later on in my program with np.resize() to make it dimension (10, 1) instead of (10). Otherwise, I am satisfied with the outcome of this test, because this shows to me that the neural network has a backpropagation algorithm that can now navigate back the neural network and output the relevant changes that need to be made to the neural network in order for a better accuracy for future iterations (epoch) of the neural network during gradient descent.

The next test I want to do with the backpropagation() method is to see how efficiently it amends the neural network weights and biases to increase the accuracy of the neural network in recognising digits.

To perform this test I will need a method that can iterate through training MNIST data and feed it into the neural network for training. The backpropagation algorithm should then update the weights and biases of the neural network based on the training data. This is called gradient descent (or Stochastic gradient descent) for which I have made an algorithm in Section 2.2.1.

Before gradient descent, I need to create a simple method that updates weights and biases based on backpropagation output. In my pseudocode for neural network algorithms in Section 2.2.1, I put this feature in my gradient descent method (gradient_descent()) but for modularity and individual testing I have separated this out to standalone method called update_weights_biases():

My project: Handwriting recognition research project

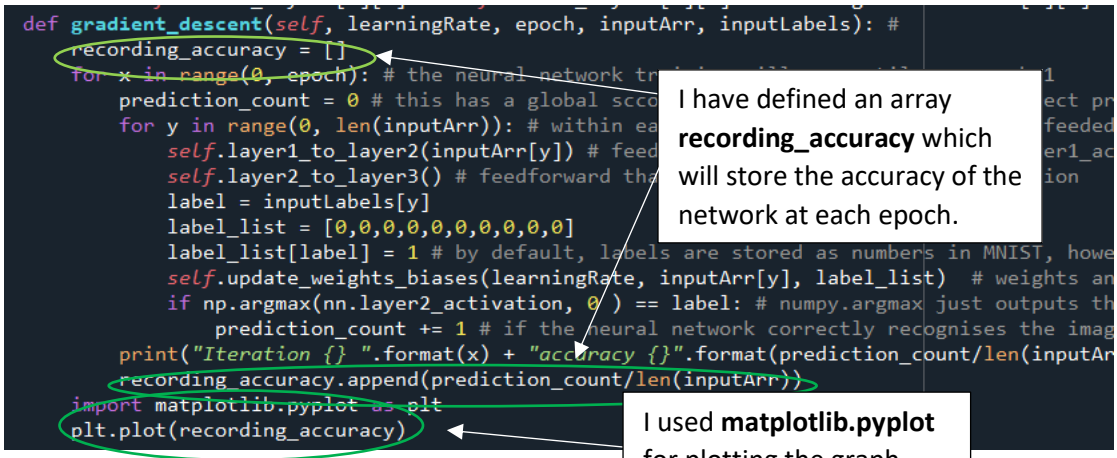
```
def update_weights_biases(self,learningRate, inputArr, inputLabel):
    dWeights1, dWeights2, dBiases1, dBiases2 = self.backpropagation(inputArr, inputLabel) # backpropagation outputs values of increment/decrement of weights of layers 2 and 3 and biases of layers 2 and 3 so the error margin can be minimised
    for x in range(0, len(self.weights_layer1)): # weights_layer1 has dimensions (10, 784) and each of the 784 elements of all 10 nodes must be updated so two loops are required, one to visit each node and one to visit all 784 elements of each node.
        for y in range(0, len(self.weights_layer1[x])):
            self.weights_layer1[x][y] = self.weights_layer1[x][y] + learningRate*dWeights1[x][y] # each element of weights_layer1 is nudged by a dWeights1 (which is multiplied by a learning rate)
    for x in range(0, len(self.biases_layer1)):
        self.biases_layer1[x][0] = self.biases_layer1[x][0] + learningRate*dBiases1[x] # each bias in biases_layer1 is nudged to minimize error margin
    for x in range(0, len(self.weights_layer2)):
        for y in range(0, len(self.weights_layer2[x])):
            self.weights_layer2[x][y] += self.weights_layer2[x][y] + learningRate*dWeights2[x][y]  # each element of weights_layer2 is nudged by a dWeights1 (which is multiplied by a learning rate)
    #self.biases_layer2 = self.biases_layer2 - alpha * dBiasess2
    for x in range(0, len(self.biases_layer2)):
        self.biases_layer2[x][0] = self.biases_layer2[x][0] + learningRate*dBiases2[x][0] # each bias in biases_layer1 is nudged to minimize error margin
def gradient_descent(self, learningRate, epoch, inputArr, inputLabels):
```

My code for gradient descent is below:

```
def gradient_descent(self, learningRate, epoch, inputArr, inputLabels): #
    for x in range(0, epoch): # the neural network training will run until x = epoch-1
        prediction_count = 0 # this has a global sccope, it stores the number of correct predictions and will be used for determining accuracy
        for y in range(0, len(inputArr)): # within each epoch, the neural network is feeded with MNIST image data
            self.layer1_to_layer2(inputArr[y]) # feedforward that determines self.layer1_activation
            self.layer2_to_layer3() # feedforward that determines self.layer2_activation
            label = inputLabels[y]
            label_list = [0,0,0,0,0,0,0,0,0,0]
            label_list[label] = 1 # by default, labels are stored as numbers in MNIST, however for my neural network, I need to convert it to a list where the index of label number is equal to 1
            self.update_weights_biases(learningRate, inputArr[y], label_list)  # weights and biases are updated based on MNIST image array
            if np.argmax(nn.layer2_activation, 0 ) == label: # numpy.argmax just outputs the index of the highest probability index in layer2_activation
                prediction_count += 1 # if the neural network correctly recognises the image, increment the prediction count
        print("Iteration {} ".format(x) + "accuracy {}".format(prediction_count/len(inputArr))) # at the end of each epoch, the accuracy of the neural network is output
```

Graphing accuracy:

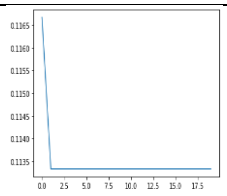
For my testing, I will need to visualise the results of my neural network so I can see the general trend (gradient) of the accuracy of the neural network. This is the amendment I have made to my code to facilitate graphing:



Test	Input data/input code	Desired result	Actual result	Pass/ Fail	Explanation/Justification
Running gradient descent and observing whether the accuracy of the neural network increases	openFile = open("MNIST_resized_ready_for_nn") import json input1 = json.load(openFile) print("program started")	The general trend of accuracy of the neural network increases with average accuracy above 70%	c:\users\sahil\desktop\computing project\handwriting-recognition-ann-master\mnist_doing_stuff\test_strictly.py:35: RuntimeWarning: overflow encountered in exp exp_total += round(np.exp(x[index]),	Fail	I have a Runtime Warning which is not an error but it sets all the values of the weights and biases as NaN values, meaning my neural network cannot perform any meaningful mathematical

with the number of epoch or not.	input_array_total = input1["training_resized"][:300] label_total = input1["training_label"] nn = Network() nn.gradient_descent(0.9, 20, input_array_total, label_total)				calculations and therefore would output erroneous results. The likely cause of this is my **softmax()** function where I have to division by e^sum which can be 0, and division by 0 will produce an error. Another likely cause of the error can be high output values for euler's number (for example, if x = 25, e^x = 72,004,899,337.3858725241613514661 26 so values of e^x can get high really quickly), which can be hard to store and do calculations with, since I am using NumPy arrays with float64 elements, it can easily overflow (i.e. values can go over 64 bits)

To solve a Runtime Warning, I visited machine learning - Common causes of nans during training of neural networks - Stack Overflow and one of the solutions that I found was reducing the learning rate of the gradient descent. If I reduce the learning rate of the neural network, my values for weights will be small, so e^x will generally output smaller values (below 64 bits). I will try learning_rate = 0.01

Test	Input data/input code	Desired result	Actual result	Pass/Fail	Explanation/Justification
Running gradient descent and observing whether the	openFile = open("MNIST_resized_ready_for_nn") import json	The general trend of accuracy of the neural network increases.		Pass	It has passed my testing because the program generates accuracy and runs for 20

accuracy of the neural network increases with the number of epoch or not.	input1 = json.load(openFile) print("program started") input_array_total = input1["training_resized"][:300] label_total = input1["training_label"] nn = Network() nn.gradient_descent(0.01, 20, input_array_total, label_total)				epochs. But I am not satisfies, because the accuracy has went down between epoch 1 and 2 and then stayed the same between epoch 2 and epoch 20. This is definitely erroneous to me and I will have to investigate this further.

Section 3.1.5) Review

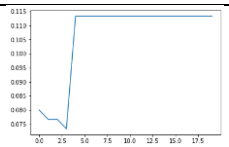
Review of Stage 1

Work I have done

In Stage 1, I have made a neural network from scratch for MNIST dataset (that can be used with EMNIST as well, which is what I need for letter recognition). In stage 1, I made:

1. Initial weights and biases which will be adjusted later (based on input data)
2. Initial weights and biases of layers 2 and 3, which will be adjusted later (based on input data)
3. Feedforward (**layer1_to_layer2()** and **layer_2_to_layer3()** methods) which perform activations (sigmoid and softmax) on weights of layer 2 and 3 and input data
4. Backpropagation where weights and biases are updated by traversing back through the neural network and using calculus (chain rule) in order to find optimal values for incrementing original weights and biases.

Testing of my work

Test	Input data/input code	Desired result	Actual result	Pass/Fail	Explanation/Justification
My neural network has success in recognising	openFile = open("MNIST_resized_ready_for_nn") import json	The accuracy of my neural network is above 50% (so it proves	 The accuracy is 11.3%	Fail	The backpropagation of my neural network is not doing anything meaningful to

digits/letters	input1 = json.load(openFile) print("program started") input_array_total = input1["training_resized"][:300] label_total = input1["training_label"] nn = Network() nn.gradient_descent(0.01, 20, input_array_total, label_total)	its not guesses but is actually recognising)			make the neural network more efficient. While not erroneous, my neural network is certainly not at all functionable in its role for recognising digits/letters. I do not think I can use this neural network in my project.

Has it met my key features of my solution (Section 1.5)/success criteria (Section 1.6)?

Key features (neural network)	Met/Not Met	Explanation/Justification
Make an accurate neural network from scratch that can recognise letters	Not Met	While I am satisfied with my feedforward part of my neural network, I am disappointed in the backpropagation algorithm of my neural network, due to which the neural network accuracy in recognition of letters/digits is consistently below 15% (which is on par with guessing). I don't think this is a feasible solution to implement in my interface moving forward.
Backpropagation algorithm is efficient and accurately updates the neural network	Not Met	My backpropagation algorithm does not function as intended. While it doesn't show any errors (such as Type error) it doesn't do anything to make the neural network more efficient. Because of this, my neural network is not feasible for my project and I don't think I will use it for my interface

Split the image dataset into training and test dataset to avoid overfitting	Met	At the start of stage 1, I have made sure that my MNIST dataset was split into training/test data to avoid overfitting/underfitting

Steps to improve it

To improve my neural network, I have decided to use the Tensorflow module and its vast reserve of resources and methods to make a CNN that is accurate, reliable and matches my key features' points closely (**Section 1.5**). Even though in **Section 1.4.2** I said that I will not use Tensorflow because it is too easy and straightforward, I now know that making a neural network from scratch is indeed very hard and I need to use Tensorflow and its amazing and accurate algorithms in order to yield accuracy and quality for my project.

I will redo stage 1 with Tensorflow and reassess the stage once I have finished it.

Section 3.2) Stage 1 redone using TensorFlow

Section 3.2.1) Loading and changing EMNIST for neural network

Since I have a time constraint already, and I am relatively unfamiliar with Tensorflow, I will learn and use parts of code from the website [EMNIST Letter Dataset 97.9%:acc & val_acc: 91.78% | Kaggle](), where the author explains how to make a basic neural network in Tensorflow for the EMNIST dataset (the letter dataset) and it has a good accuracy as well.

Before we make the neural network, we must import the EMNIST dataset. For my program I have downloaded the Balanced EMNIST dataset, which are CSV files containing EMNIST testing and training data, and has 46 output classes. I used the pandas module to read the .csv files.

```
import pandas as pd
training_data = pd.read_csv("emnist-letters-train.csv") # reading training data from csv file using pandas module
testing_data = pd.read_csv("emnist-letters-test.csv") # reading testing data from csv file using pandas module
```

However, as I executed the code above, I noticed that the dimensions of **training_data** and **testing_data** are not in the standard (x, 784) or (x ,28, 28) format (where x is any arbitrary integer).

testing_data	DataFrame	(14799, 785)	Column names: 1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.10, ...
training_data	DataFrame	(88799, 785)	Column names: 23, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.10 ...

One of the dimensions is 785 as opposed to the standard 784 (which are the number of pixels in a 28x28 image format).

My project: Handwriting recognition research project

Since testing_data, and training_data aren't NumPy arrays yet, I cannot check its elements and indexes so I will use np.array() to convert testing_data, and training_data into NumPy arrays.

```
train_data = np.array(training_data) # converts training_data to numpy arrays so elements can be called using its indexes
test_data = np.array(testing_data) # converts testing_data to numpy arrays so elements can be called using its indexes
```

After converting testing_data, and training_data into NumPy arrays (train_data, and test_data) I noticed that the first index of each element of train_data, and test_data contains the label (i.e. corresponding output) to the respective image data.

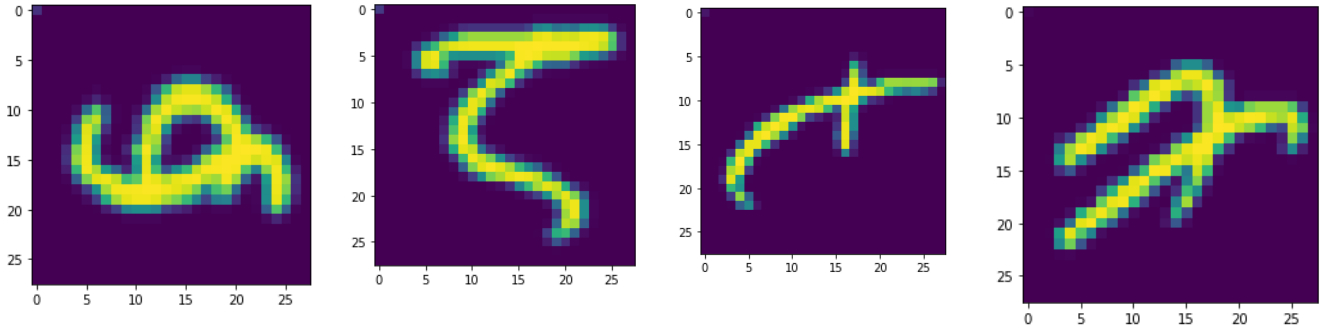
These are the labels of **train_data**, and I am sure that the structure of **test_data** is similar

```
train_labels = []
for x in range(0, len(train_data)):
    train_labels.append(train_data[x][0]) # I add the first element of train_data[x] (the label) to train_labels
test_labels = []
for x in range(0, len(test_data)):
    test_labels.append(test_data[x][0]) # I add the first element of test_data[x] (the label) to test_labels
```

Using the above information, I can store the labels of **train_data** and **test_data** in new arrays **train_labels**, and **test_labels**

My project: Handwriting recognition research project

Now that I have sorted out the training and testing arrays for EMNIST, I must rotate (90 degrees anti-clockwise) and flip the image. This is because, according to [EMNIST Classification – Weights & Biases (wandb.ai)] EMNIST dataset is inverted horizontally and rotated 90 degrees clockwise. This is what the EMNIST dataset looks like:



This is my code to rotate each of the images anticlockwise and flip it and save it back to EMNIST, so the neural network can train on letters that are oriented the correct way.

```
for x in range(0, len(train_data)):
    img = np.resize(train_data[x][1:], (28,28)) # I have to reshape the EMNIST data from (784) to (28, 28) for np.fliplr() and np.rot90() to work
    img_flip = np.fliplr(img) # flips image
    img_rotate = np.rot90(img_flip) # rotates image by 90 degrees
    train_data[x] = img_final # save it back as an element of train_data
for x in range(0, len(test_data)):
    img = np.resize(test_data[x][1:], (28,28)) # I have to reshape the EMNIST data from (784) to (28, 28) for np.fliplr() and np.rot90() to work
    img_flip = np.fliplr(img) # flips image
    img_rotate = np.rot90(img_flip) # rotates image by 90 degrees
    test_data[x] = img_final # save it back as an element of test_data
```

However, I get an error because I cannot save an element as a new array that is of a different shape.

```
    train_data[x] = img_final

ValueError: could not broadcast input array
from shape (28,28) into shape (785)
```

This is my amended code:

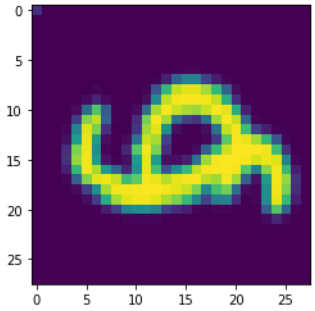
I will store the rotated and flipped images into new arrays **new_train_data** and **new_test_data**.

```
new_train_data = []
for x in range(0, len(train_data)):
    img = np.resize(train_data[x][1:], (28,28)) # I have to reshape the EMNIST data from (784) to (28, 28) for np.fliplr() and np.rot90() to work
    img_flip = np.fliplr(img) # flips image
    img_rotate = np.rot90(img_flip) # rotates image by 90 degrees
    new_train_data.append(img_rotate) # save it back as an element of train_data
new_test_data = []
for x in range(0, len(test_data)):
    img = np.resize(test_data[x][1:], (28,28)) # I have to reshape the EMNIST data from (784) to (28, 28) for np.fliplr() and np.rot90() to work
    img_flip = np.fliplr(img) # flips image
    img_rotate = np.rot90(img_flip) # rotates image by 90 degrees
    new_test_data.append(img_rotate) # save it back as an element of test_data
```

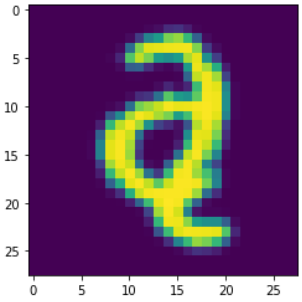
The results of the transformation of EMNIST images:

My project: Handwriting recognition research project

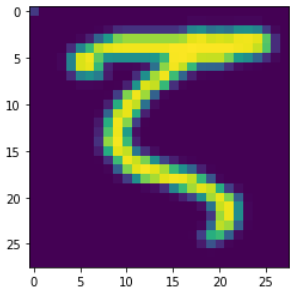
Before:



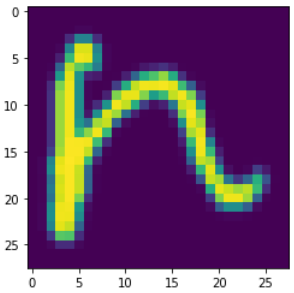
After:



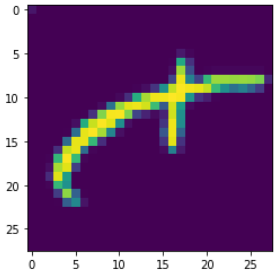
Before:



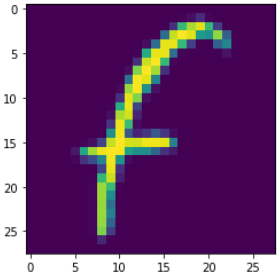
After:



Before:



After:



Section 3.2.2) Creating a neural network using TensorFlow

Creating my neural network:

To create my neural network, I just copied the structure of the neural network from EMNIST Letter Dataset 97.9%:acc & val_acc: 91.78% | Kaggle. This is because, since I don't have much knowledge for TensorFlow, I will struggle to make it without any external support. I have also failed in my previous attempt at making my neural network, and I need to remake it which is time-consuming, so I think it is better for me if I can use a template of an already successful neural network because it will aid me in my handwriting recognition.

This is the input layer which only accepts input data in the dimensions (28, 28, 1) so I will have to use NumPy resize() method to reshape my training and testing dataset to facilitate this.

My project: Handwriting recognition research project

```
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32,3, input_shape=(28,28, 1)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(input_shape=(28,28, 1)),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(47,activation='softmax')
])
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

I have made the final **Dense** layer to have 47 nodes because it is the output layer where the output of probabilities of prediction for an image will be output and there are 47 output classes for my **EMNIST Balanced dataset.**

nn.compile() method just defines the optimizer (i.e. something to reduce the cost of the network (similar to backpropagation)), and the cost(loss) function (difference between the predicted and desired outcome). The **ADAM (Adaptive Moment Estimation)** optimizer is algorithm used to reduce the cost function, the author of [EMNIST Letter Dataset 97.9%:acc & val_acc: 91.78% | Kaggle] used a different optimizer "rmsprop", but I used ADAM because it is a standard and reliable optimizer that I have seen people use alot. **SparseCategoricalCrossentropy** is a type of cost/loss calculation algorithm. I used **SparseCategoricalCrossentropy** as opposed to the author (of the Kaggle code) implementation (**categorical_crossentropy**) because they are using one hot arrays as labels whereas I am using an array with numeric labels.

CNN training by

```
new_train_data = np.array(new_train_data, dtype = np.float64) # I will convert new_train_data to a NumPy array as tensorflow CNN can only accept NumPy arrays
new_train_data = np.resize(new_train_data, (len(new_train_data), 28, 28, 1)) # since the input for the sequential CNN is (28, 28, 1) we need to resize the new_train_data to facilitate that
train_labels = np.resize(train_labels, (len(train_labels), 1)) # the training label array is being formatted from list of number labels to list of arrays containging number labels
new_test_data = np.array(new_test_data, dtype = np.float64) # I will convert new_train_data to a NumPy array as tensorflow CNN can only accept NumPy arrays
new_test_data = np.resize(new_test_data, (len(new_test_data), 28, 28, 1)) # since the input for the sequential CNN is (28, 28, 1) we need to resize the new_test_data to facilitate that
test_labels = np.resize(test_labels, (len(test_labels), 1)) # the testing label array is being formatted from list of number labels to list of arrays containging number labels
train_labels = np.array(train_labels, np.uint8)
test_labels = np.array(test_labels, np.uint8)
history = model.fit(new_train_data, train_labels,  epochs = 50, validation_data=(new_test_data, test_labels))
model.save('final_CNN')
```

Saves model for future use

nn.fit() will be used to train the neural network with training data (**new_train_data**) and training labels (**train_labels**). It will run for 50 epochs. I have also added test data (**new_test_data** and **test_label**) for validation of the network after each iteration.

Since TensorFlow only accepts NumPy arrays for training, I had to use np.array() and dtype = np.float64 (for training data) and dtype = np.uint8 (for training labels) so it can be acceptable for **model.fit()**

My final code:

My project: Handwriting recognition research project

```python
import pandas as pd
import numpy as np
training_data = pd.read_csv("emnist-balanced-train.csv") # reading training data from csv file using pandas module
testing_data = pd.read_csv("emnist-balanced-test.csv") # reading testing data from csv file using pandas module
#size of training_data is (88799, 785)
#size of testing_data is (14799, 785)
train_data = np.array(training_data, dtype = np.float64) # converts training_data to numpy arrays so elements can be c
test_data = np.array(testing_data, dtype = np.float64) # converts testing_data to numpy arrays so elements can be call
train_labels = []
for x in range(0, len(train_data)):
    train_labels.append(train_data[x][0]) # I add the first element of train_data[x] (the label) to train_labels
test_labels = []
for x in range(0, len(test_data)):
    test_labels.append(test_data[x][0]) # I add the first element of test_data[x] (the label) to test_labels
train_data = train_data/255.0
test_data = test_data/255.0
new_train_data = []
for x in range(0, len(train_data)):
    img = np.resize(train_data[x][1:], (28,28)) # I have to reshape the EMNIST data from (784) to (28, 28) for np.fliplr() and np.rot90() to work
    img_flip = np.fliplr(img) # flips image
    img_rotate = np.rot90(img_flip) # rotates image by 90 degrees
    new_train_data.append(img_rotate) # save it back as an element of train_data
new_test_data = []
for x in range(0, len(test_data)):
    img = np.resize(test_data[x][1:], (28,28)) # I have to reshape the EMNIST data from (784) to (28, 28) for np.fliplr() and np.rot90() to work
    img_flip = np.fliplr(img) # flips image
    img_rotate = np.rot90(img_flip) # rotates image by 90 degrees
    new_test_data.append(img_rotate) # save it back as an element of test_data
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32,3, input_shape=(28,28, 1)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(input_shape=(28,28, 1)),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(47,activation='softmax')
])
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
new_train_data = np.array(new_train_data, dtype = np.float64) # I will convert new_train_data to a NumPy array as tensorflow CNN can only accept NumPy arrays
new_train_data = np.resize(new_train_data, (len(new_train_data), 28, 28, 1)) # since the input for the sequential CNN is (28, 28, 1) we need to resize the new_train_data to facilitate that
train_labels = np.resize(train_labels, (len(train_labels), 1)) # the training label array is being formatted from list of number labels to list of arrays containging number labels
new_test_data = np.array(new_test_data, dtype = np.float64) # I will convert new_train_data to a NumPy array as tensorflow CNN can only accept NumPy arrays
new_test_data = np.resize(new_test_data, (len(new_test_data), 28, 28, 1)) # since the input for the sequential CNN is (28, 28, 1) we need to resize the new_test_data to facilitate that
test_labels = np.resize(test_labels, (len(test_labels), 1)) # the testing label array is being formatted from list of number labels to list of arrays containging number labels
train_labels = np.array(train_labels, np.uint8)
test_labels = np.array(test_labels, np.uint8)
history = model.fit(new_train_data, train_labels,  epochs = 50, validation_data=(new_test_data, test_labels))
model.save("final_CNN")
```

I have additionally normalised the test and training data so the dataset images are clearer (less blurry)

My output:

```
3525/3525 [==============================] - 68s 19ms/step - loss: 0.0934 - accuracy: 0.9685 - val_loss: 1.6914 - val_accuracy: 0.8413
Epoch 32/50
3525/3525 [==============================] - 68s 19ms/step - loss: 0.0889 - accuracy: 0.9690 - val_loss: 1.5920 - val_accuracy: 0.8365
Epoch 33/50
3525/3525 [==============================] - 69s 19ms/step - loss: 0.0930 - accuracy: 0.9693 - val_loss: 1.6539 - val_accuracy: 0.8388
Epoch 34/50
3525/3525 [==============================] - 68s 19ms/step - loss: 0.0904 - accuracy: 0.9706 - val_loss: 1.6986 - val_accuracy: 0.8427
Epoch 35/50
3525/3525 [==============================] - 69s 20ms/step - loss: 0.0899 - accuracy: 0.9710 - val_loss: 1.7596 - val_accuracy: 0.8387
Epoch 36/50
3525/3525 [==============================] - 70s 20ms/step - loss: 0.0886 - accuracy: 0.9722 - val_loss: 1.8645 - val_accuracy: 0.8342
Epoch 37/50
3525/3525 [==============================] - 67s 19ms/step - loss: 0.0875 - accuracy: 0.9720 - val_loss: 1.8900 - val_accuracy: 0.8389
Epoch 38/50
3525/3525 [==============================] - 67s 19ms/step - loss: 0.0853 - accuracy: 0.9730 - val_loss: 1.9641 - val_accuracy: 0.8402
Epoch 39/50
3525/3525 [==============================] - 67s 19ms/step - loss: 0.0832 - accuracy: 0.9735 - val_loss: 2.0824 - val_accuracy: 0.8392
Epoch 40/50
3525/3525 [==============================] - 67s 19ms/step - loss: 0.0869 - accuracy: 0.9735 - val_loss: 2.0365 - val_accuracy: 0.8381
Epoch 41/50
3525/3525 [==============================] - 69s 20ms/step - loss: 0.0839 - accuracy: 0.9746 - val_loss: 2.1299 - val_accuracy: 0.8394
Epoch 42/50
3525/3525 [==============================] - 67s 19ms/step - loss: 0.0866 - accuracy: 0.9746 - val_loss: 2.1547 - val_accuracy: 0.8366
Epoch 43/50
3525/3525 [==============================] - 70s 20ms/step - loss: 0.0815 - accuracy: 0.9758 - val_loss: 2.1012 - val_accuracy: 0.8367
Epoch 44/50
3525/3525 [==============================] - 86s 24ms/step - loss: 0.0838 - accuracy: 0.9757 - val_loss: 2.1845 - val_accuracy: 0.8380
Epoch 45/50
3525/3525 [==============================] - 70s 20ms/step - loss: 0.0761 - accuracy: 0.9773 - val_loss: 2.3020 - val_accuracy: 0.8376
Epoch 46/50
3525/3525 [==============================] - 68s 19ms/step - loss: 0.0775 - accuracy: 0.9773 - val_loss: 2.4447 - val_accuracy: 0.8386
Epoch 47/50
3525/3525 [==============================] - 71s 20ms/step - loss: 0.0842 - accuracy: 0.9767 - val_loss: 2.3794 - val_accuracy: 0.8387
Epoch 48/50
3525/3525 [==============================] - 74s 21ms/step - loss: 0.0781 - accuracy: 0.9782 - val_loss: 2.4274 - val_accuracy: 0.8347
Epoch 49/50
3525/3525 [==============================] - 72s 20ms/step - loss: 0.0768 - accuracy: 0.9780 - val_loss: 2.5302 - val_accuracy: 0.8362
Epoch 50/50
3525/3525 [==============================] - 70s 20ms/step - loss: 0.0800 - accuracy: 0.9780 - val_loss: 2.6891 - val_accuracy: 0.8360

2022-04-14 17:20:39.596486: W tensorflow/python/util/util.cc:348] Sets are not currently considered sequences, but this may change in the future, so consider avoiding using them.
INFO:tensorflow:Assets written to: final_CNN\assets
```

The **val_accuracy** refers to the accuracy of the neural network in recognising letters from **new_test_data**, which is very high (consistently in the region of **83%**)

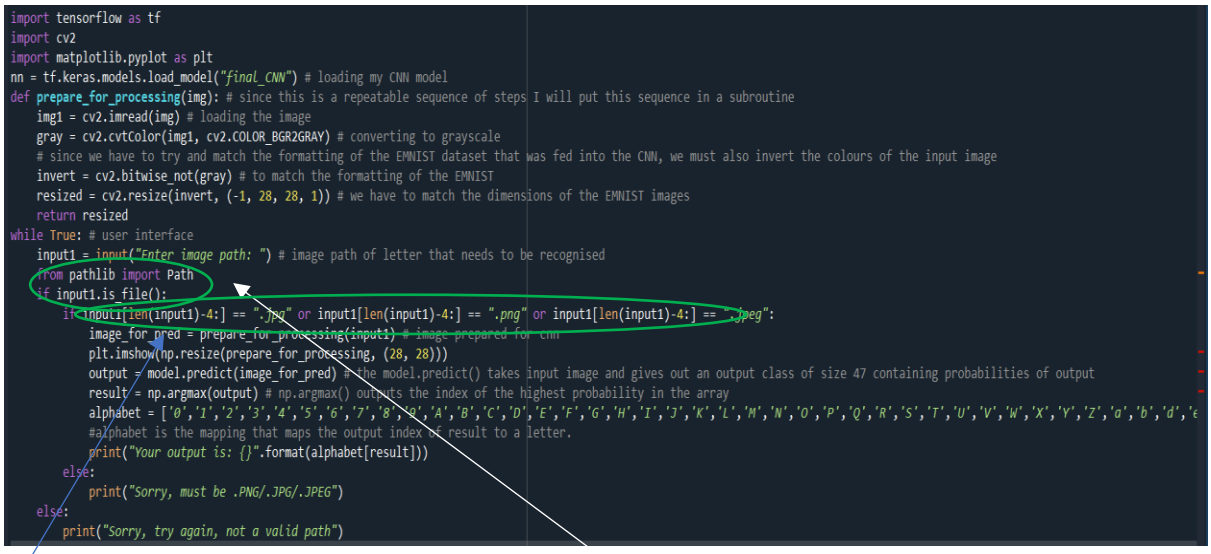
The accuracy of the neural network is consistently **97.8%** which is very encouraging!

The high **accuracy** and **val_accuracy** of the neural network indicate to me that the neural network is very reliable and has a low chance of outputting erroneous results.

My project: Handwriting recognition research project

Section 3.2.3) Testing my neural network (and validation of input):

To test my neural network, I first need to make a user interface that is validated for the format of image from the user (i.e., if it is a .PNG/.JPEG/.JPG then allow it to be input otherwise don't). The reason I am so adamant about only allowing .PNG/.JPEG/.JPG is because they are the only reliable image formats that cannot cause any erroneous results directly. Also, I need some way of verifying that only images are being input for neural network and not other formats like .txt, .gif, etc, which will cause errors in my program.



```
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
nn = tf.keras.models.load_model("final_CNN") # loading my CNN model
def prepare_for_processing(img): # since this is a repeatable sequence of steps I will put this sequence in a subroutine
    img1 = cv2.imread(img) # loading the image
    gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY) # converting to grayscale
    # since we have to try and match the formatting of the EMNIST dataset that was fed into the CNN, we must also invert the colours of the input image
    invert = cv2.bitwise_not(gray) # to match the formatting of the EMNIST
    resized = cv2.resize(invert, (-1, 28, 28, 1)) # we have to match the dimensions of the EMNIST images
    return resized
while True: # user interface
    input1 = input("Enter image path: ") # image path of letter that needs to be recognised
    from pathlib import Path
    if input1.is_file():
        if input1[len(input1)-4:] == ".jpg" or input1[len(input1)-4:] == ".png" or input1[len(input1)-4:] == ".jpeg":
            image_for_pred = prepare_for_processing(input1) # image prepared for CNN
            plt.imshow(np.resize(prepare_for_processing, (28, 28)))
            output = model.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 containing probabilities of output
            result = np.argmax(output) # np.argmax() outputs the index of the highest probability in the array
            alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','d','e
            #alphabet is the mapping that maps the output index of result to a letter.
            print("Your output is: {}".format(alphabet[result]))
        else:
            print("Sorry, must be .PNG/.JPG/.JPEG")
    else:
        print("Sorry, try again, not a valid path")
```

This is my second user validation, where I check whether the verified file path is a .JPG/.JPEG/.PNG image or not. If it is (i.e. if the last 4 letters of the user input is ".jpg", ".jpeg", or ".png") then allow the CNN recognition to continue, otherwise the user has to re-enter their path.

This is my first user validation. I use .is_file() method from pathlib module to verify whether the file path being input is valid. If it is then .if_file() outputs Boolean True and the CNN recognition takes place, or else false, and the user is told to retry.

Why I need validation for this user interface (explanation/justification)?

This user interface is for internal testing during the project (for example for testing the effects of increasing thickness of images on recognition of letters, etc) so the client will never actually use this exact interface, however, to minimise errors when passing user input through my program, I must verify that the user input passes the 2 tests above. If the validity of the file path is not verified, then I can get errors for **cv2.imread()** and if the image format of the input isn't verified then I can get error for **model.predict()** as it is not in the format TensorFlow wants.

My code (same screenshot as above but this time I will do analysis on image processing):

My project: Handwriting recognition research project

```python
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
nn = tf.keras.models.load_model("final_CNN") # loading my CNN model
def prepare_for_processing(img): # since this is a repeatable sequence of steps I will put this sequence in a subroutine
    img1 = cv2.imread(img) # loading the image
    gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY) # converting to grayscale
    # since we have to try and match the formatting of the EMNIST dataset that was fed into the CNN, we must also invert the colours of the input image
    invert = cv2.bitwise_not(gray) # to match the formatting of the EMNIST
    resized = cv2.resize(invert, (-1, 28, 28, 1)) # we have to match the dimensions of the EMNIST images
    return resized
while True: # user interface
    input1 = input("Enter image path: ") # image path of letter that needs to be recognised
    from pathlib import Path
    if input1.is_file():
        if input1[len(input1)-4:] == ".jpg" or input1[len(input1)-4:] == ".png" or input1[len(input1)-4:] == ".jpeg":
            image_for_pred = prepare_for_processing(input1) # image prepared for cnn
            plt.imshow(np.resize(prepare_for_processing, (28, 28)))
            output = model.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 containing probabilities of output
            result = np.argmax(output) # np.argmax() outputs the index of the highest probability in the array
            alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','d','e
            #alphabet is the mapping that maps the output index of result to a letter.
            print("Your output is: {}".format(alphabet[result]))
        else:
            print("Sorry, must be .PNG/.JPG/.JPEG")
    else:
        print("Sorry, try again, not a valid path")
```

The output of **model.predict()** (**output**) is an array with length 47 which contain softmax probabilities of the result.

To output the user input in a format that is acceptable for my neural network, I must use **cv2.cvtColor()** and **cv2.bitwise_not()** to convert the image to grayscale and then invert it, as all EMNIST images have a black background with white foreground. The image must also be resized to (1, 28,28, 1) because that's what the first **Dense** layer accepts

np.argmax() outputs the index with element of highest probability, the index can then be used in **alphabet** to map out the letter output.

When I executed the code, I got an error:

```
line 21, in <module>
    if input1.is_file():

AttributeError: 'str' object has no
attribute 'is_file'
```

This is because I forgot to add **Path(input1).is_file()** to line 21:

```python
from pathlib import Path
if Path(input1).is_file():
    if input1[len(input1)-4:] ==
```

I added **Path(input1).is_file()**

When I re executed the code, I got a new error, this time for line 16:

My project: Handwriting recognition research project

```
line 16, in prepare_for_processing
    resized = cv2.resize(invert, (1, 28,
28, 1)) # we have to match the dimensions
of the EMNIST images

error: OpenCV(4.5.2) :-1: error: (-5:Bad
argument) in function 'resize'
> Overload resolution failed:
>  - Can't parse 'dsize'. Expected sequence
length 2, got 4
>  - Can't parse 'dsize'. Expected sequence
length 2, got 4
```

This is the line where I had the code **resized = cv2.resize(invert, (1, 28, 28, 1))**. I realised that cv2.resize() cannot convert anything behold two parameters (e.g. (40,32)). I will need to use **np.resize()** which is what I used to make my neural network weights and biases.

Here is my amended code:

```
resized = np.resize(invert, (1, 28, 28, 1)) # we have
return resized
```

After this amendment, my code works. This is the output for an image “d.png”:

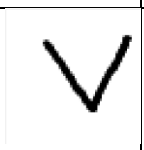
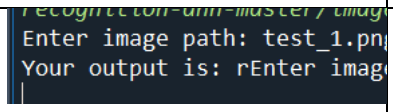
```
In [19]: runfile('C:/Users/Sahil/Desktop/Computing project/ho
recognition-ann-master/image_recog_stuff')
Enter image path: d.png
Your output is: BEnter image path: d.png
```

My final code for user interface for neural network:

```
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
import numpy as np
nn = tf.keras.models.load_model("final_CNN") # loading my CNN model
def prepare_for_processing(img): # since this is a repeatable sequence of steps I will put this sequence in a subroutine
    img1 = cv2.imread(img) # loading the image
    gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY) # converting to grayscale
    # since we have to try and match the formatting of the EMNIST dataset that was fed into the CNN, we must also invert the colours of the input image
    invert = cv2.bitwise_not(gray) # to match the formatting of the EMNIST
    resized = np.resize(invert, (1, 28, 28, 1)) # we have to match the dimensions of the EMNIST images
    return resized
while True: # user interface
    input1 = input("Enter image path: ") # image path of letter that needs to be recognised
    from pathlib import Path
    if Path(input1).is_file():
        if input1[len(input1)-4:] == ".jpg" or input1[len(input1)-4:] == ".png" or input1[len(input1)-4:] == ".jpeg":
            image_for_pred = prepare_for_processing(input1) # image prepared for cnn
            #plt.imshow(np.resize(prepare_for_processing, (28, 28)))
            output = nn.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 containing probabilities of output
            result = np.argmax(output) # np.argmax() outputs the index of the highest probability in the array
            alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','d','e
            #alphabet is the mapping that maps the output index of result to a letter.
            print("Your output is: {}".format(alphabet[result]))
        else:
            print("Sorry, must be .PNG/.JPG/.JPEG")
    else:
        print("Sorry, try again, not a valid path")
```

Testing on neural network interface (attempt 2)

Test	Input data/input code	Desired output	Actual output	Pass/ Fail	Explanation/Justification
I want check whether my input		The digit “u” should be	master/image_recog_stuff') Enter image path: test_1.p Your output is: I	Fail	I do not have an explanation for this failed test, I presume that cv2 and my NumPy

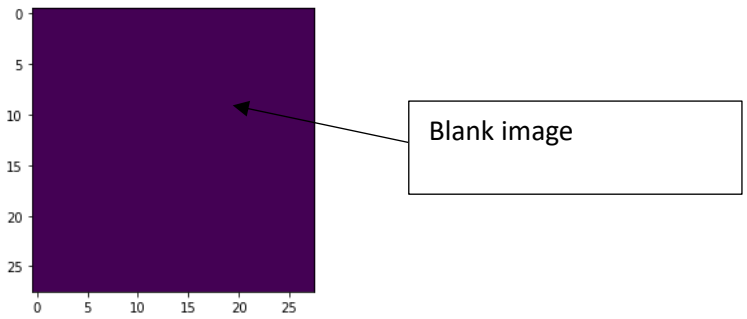
letter is recognised by my neural network or not		output by the neural network			arrays are not outputting a good image to be sent to the neural network (in my prepare_for_processing()).
I want check whether my input letter is recognised by my neural network or not		The digit "V" should be output by the neural network		Fail	I think that my prepare_for_processing() isn't outputting a clear image that can be sent the neural network. I must investigate the solution.

To diagnose the problem I put used the library matplotlib to output the image that is being sent to the neural network.

```
if input1[len(input1)-4:] == ".jpg" or input1[len(input1)-4:] == ".png" or input1[len(input1)-4:] == ".jpeg":
    image_for_pred = prepare_for_processing(input1) # image prepared for cnn
    plt.imshow(np.resize(image_for_pred, (28,28)))
    plt.show()
```

This is the input image for my program:

This is the matplotlib plot of prepare_for_processing() output (with the above image as its input):



So the root of my problem (regarding neural network recognision) is my prepare_for_processing() method.

One of the solutions to my problem is to use another library called PIL. PIL is similar to OpenCV in that it too can open images and perform grayscale and inverse algorithms. I

My project: Handwriting recognition research project

visited this link [python - Tensorflow: open a PIL.Image? - Stack Overflow](#) where I got an insight into how to use PIL. I am using PIL because during my research I have seen a lot of people use this library for TensorFlow.

This is a function I made using PIL:

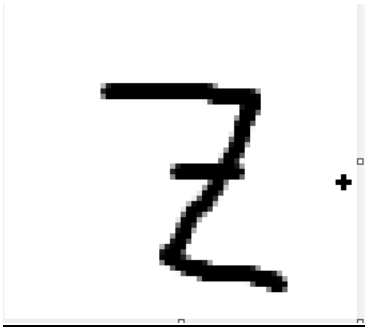
```
from PIL import Image, ImageOps
def process_letter(img_path):
    image = Image.open(img_path) # opening an image using its path
    image_resized = image.resize((28,28)) # resizing image to 28 by 28 so that inverse and grayscale operations can be performed
    image_gray = image_resized.convert("L") # converts to grayscale
    image_inverse = ImageOps.invert(image_gray) # inverts the image so the background is black and foreground is white
    numpy_image = np.array(image_inverse) # converting to numpy array for reshaping it for CNN
    final1 = numpy_image.reshape(1, 28, 28, 1) # resizing array for CNN
    final1 = final1/255 # normalising the data so it is continuous
    return final1
```

In my while loop, I changed the code to facilitate **process_letter():**

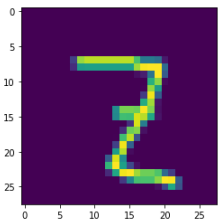
```
while True: # user interface
    input1 = input("Enter image path: ") # image path of letter that needs to be recognised
    from pathlib import Path
    if Path(input1).is_file() == True:
        if input1[len(input1)-4:] == ".jpg" or input1[len(input1)-4:] == ".png" or input1[len(input1)-4:] == ".jpeg":
            #image_for_pred = prepare_for_processing(input1) # image prepared for cnn
            image_for_pred = process_letter(input1)
            output = nn.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 cont
            result = np.argmax(output) # np.argmax() outputs the index of the highest probabilimty in the array
            alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
            #alphabet is the mapping that maps the output index of result to a letter.
            print("Your output is: {}".format(alphabet[result]))
        else:
            print("Sorry, must be .PNG/.JPG/.JPEG")
    else:
        print("Sorry, try again, not a valid path")
```

I called my **process_letter()** method here

This is the input image for my program:



This is the matplotlib plot of **process_letter()** output (with the above image as its input):



PIL generates an inverted and (28,28) dimensions NumPy array that is clean and clear

Output of the program:

My project: Handwriting recognition research project

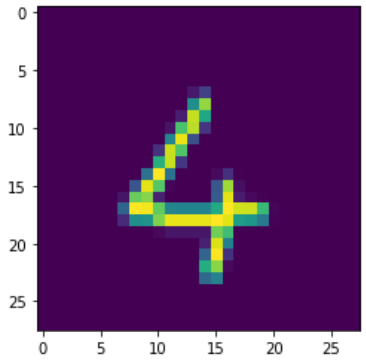
```
Enter image path: test_1.png
Your output is: ZEnter image path:
```

Correct output

I can conclude that PIL is much better for my implementation than OpenCV because with PIL, my neural network output is correct and overall very accurate. Before redoing the test on my interface, I will try my program one more time to make sure it functions correctly.

This is the input image for my program:

This is the matplotlib plot of process_letter() output (with the above image as its input):



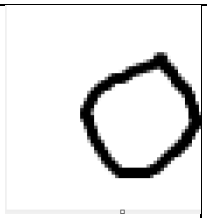
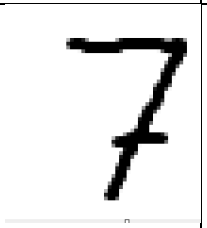
Output of the program:

```
Enter image path: test_1.png
Your output is: 4
Enter image path:
```

Correct output

Testing on neural network interface (attempt 2):

Test	Input data/input code	Desired output	Actual output	Pass/ Fail	Explanation/Justification

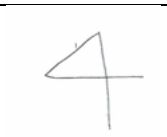
I want check whether my input letter is recognised by my neural network or not		The digit "o" or "0" should be output by the neural network	Enter image path: test_1. Your output is: O	Pass	I think that switching to PIL has improved my neural network recognition drastically which is very pleasing to see.
I want check whether my input letter is recognised by my neural network or not		The digit "V" should be output by the neural network	Enter image path: test_1. Your output is: 7 Enter image path:	Pass	

Section 3.2.4) Review

Work I have done

1. In Stage 1, I have made a neural network for the EMNIST dataset. It is very accurate with input data (handwritten text) and has an accuracy (during training) of 97%.

Using test data from Design Section Testing (Section 2.7.1):

Test	Input data/input code	Desired output	Actual output	Pass/Fail	Explanation
I will do a baseline test where I input a perfect noiseless image with the letter		I expect the neural network to correctly recognise the letter in the image (recognise it as the digit 4)	The output is 4	Pass	This doesn't surprise me, because I used similar type of data before, and my program works perfectly on those

My project: Handwriting recognition research project

I will do a another baseline test where I input a perfectly drawn letter (also with no noise)		I expect the neural network to correctly recognise the letter in the image(recognise it as the digit 9)	The output is 9	Pass	
I will do a test where the input letter is drawn at an angle to observe how the neural network responds to this scenario.		I hope that the neural network recognises this letter (H) correctly because the shape (even though it is at an angle) is identical to EMNIST images of H.	The output is H	Pass	I wasn't expecting a correct answer so I am very surprised but happy with the result.
I will input a noisy image to see what effect noise has on a neural network		I do not have high hopes for recognition of the letter in this scenario, I just want to verify how the neural network deals with noise	The output is U	Fail	Since there is so much noise in the image, I expected an erroneous result
I will deliberately enter a badly drawn letter and see how it influences my neural network		I expect erroneous output for this one because the neural network is not trained to recognise foul or bad looking letters	The output is V	Fail	The image is faulty so I wasn't expecting a correct neural network answer

Has it met my key features of my solution (Section 1.5)/success criteria (Section 1.6)?

Key features (neural network)	Met/Not Met	Explanation/Justification
Make an accurate neural network from scratch that can recognise letters	Met	TensorFlow is a great module where training a CNN is quick to implement, while also being accurate. In my screenshots above, I

		have shown that the accuracy of my neural network is over 97% (in 50 epochs) this means my CNN is indeed very accurate and ready implement further on in my project.
Backpropagation algorithm is efficient and accurately updates the neural network	Met	In my **model.compile()**, one of the parameters I needed to enter was what optimizer I will use. Optimizers are very similar to backpropagation because both are used to reduce the cost/loss in a neural network. Backpropagation specifically goes back through the neural network, however, optimizers don't necessarily have to. In my neural network I used the **ADAM (Adaptive Moment Estimation)** optimizer, which is very useful and generates a neural network with very high accuracy.
Split the image dataset into training and test dataset to avoid overfitting	Met	My dataset was already split into training and testing datasets when I downloaded the .csv files for my program.

Any improvements I can do?

I think there is still a huge room for improvement in my neural network. I haven't tested my neural network as thoroughly as I could have and I haven't retrained my model several times using several other implementations of optimizers, loss functions, layers, etc., to increase the accuracy of the network and make it more intelligent.

My reason for not spending enough time on the neural network is that I still have to work on other aspects of the program such as making the GUI, image processing algorithms, etc. I already spent a lot of my time attempting to make a neural network from scratch (which failed) and I think I should allocate more of my time to other aspects of the project, so I do not encounter problems due to time restraints.

Section 3.3) Stage 2 – Image processing

Section 3.3.1) Image preprocessing

My project: Handwriting recognition research project

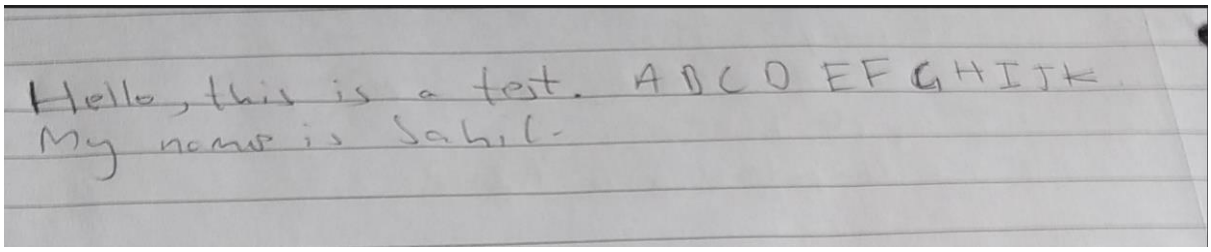
Now that I have finished building my neural network (CNN), I can now work on processing input images that the user may choose to feed into my neural network.

The first program that I want to make in this stage is for denoising (or cleaning) the input image. For this I will use a particular method in OpenCV library called Thresholding. As I have explained in **Section 2.2.2) Image processing,** that thresholding is a method in OpenCV that involves comparing pixel values of each image against a user specified threshold value If the pixel value is smaller, then it is set to 0, otherwise it is set to a maximum value (255 because thresholding only uses grayscale images). This is binary thresholding. There are also other types of thresholding that I would like to experiment with such as truncation (where if the pixel value is greater than the threshold, then the pixel value is truncated to the threshold value) and ToZero (it is similar to Binary threshold because it sets the pixel value to 0 for every pixel value that is smaller than the threshold, otherwise the pixel remains the same value).

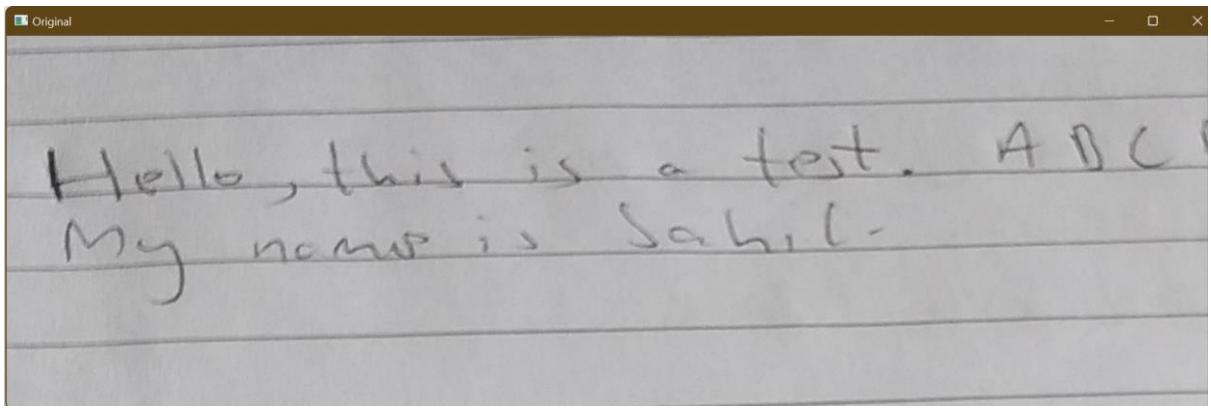
My code:

```
import cv2
imageOpen = cv2.imread("input_for_doc.jpg") # read image
imageGray = cv2.cvtColor(imageOpen, cv2.COLOR_BGR2GRAY) # converting image to grayscale as thresholding only accepts grayscale images
ret, thresh = cv2.threshold(imageGray, 100, 255, cv2.THRESH_BINARY) # Binary thresholding, if pixel value is over 100, then set it to 255, otherwise 0
ret, thresh1 = cv2.threshold(imageGray, 100, 255, cv2.THRESH_TRUNC) # Truncation thresholding, if pixel value is over 100, then set it to 255
ret2, thresh2 = cv2.threshold(imageGray, 100, 255, cv2.THRESH_TOZERO) # TO Zero thresholding, if pixel value is under 100, then set it to 0
cv2.imshow("Original", imageOpen) #showing original image for check
cv2.imshow("Grayscale", imageGray) # showing grayscale image
cv2.imshow("Binary", thresh) # showing results of Binary threshold
cv2.imshow("Truncation", thresh1) # showing results of Truncation threshold
cv2.imshow("To Zero", thresh2) # showing results of To Zero threshold

cv2.waitKey(1)
```

My original image for thresholding (**"input_for_doc.jpg"**):



When I executed **cv2. imshow("Original", imageOpen)**, my original image is cut from the left (as you can see below), so I cannot make a fair assessment of thresholding, as I cannot see the whole image.



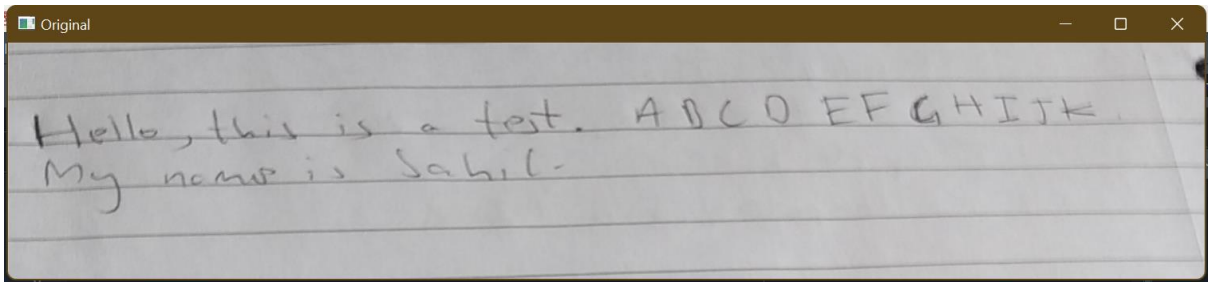
My project: Handwriting recognition research project

Therefore, I amended my code to fit a resized image (reduced size of image by a scale factor 2).

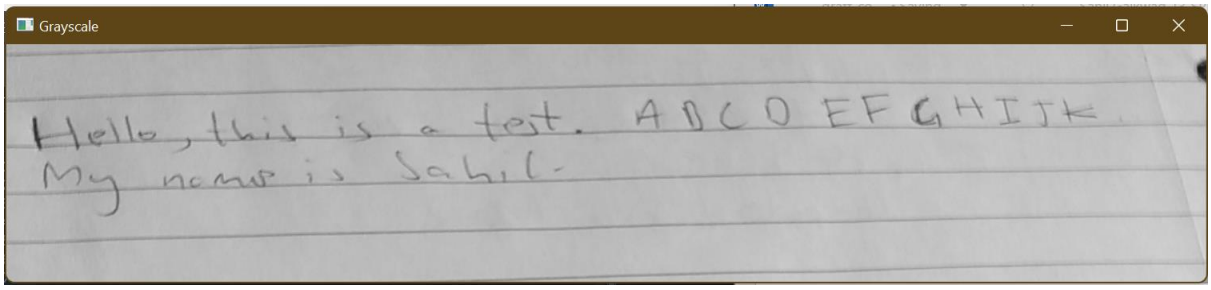
I added this code to resize image

```python
import cv2
imageOpen = cv2.imread("input_for_doc.jpg") # read image
width = int(imageOpen.shape[1]*50/100) # width of image is reduced by scale factor 2
height = int(imageOpen.shape[0]*50/100) # height of image is reduced by scale factor 2
size1 = (width,height)
imageResize = cv2.resize(imageOpen, size1) # cv2 method resize() resizes the original image
imageGray = cv2.cvtColor(imageResize, cv2.COLOR_BGR2GRAY) # converting image to grayscale as thresholding only accepts grayscale images
ret, thresh = cv2.threshold(imageGray, 100, 255, cv2.THRESH_BINARY) # Binary thresholding, if pixel value is over 100, then set it to 255, otherwise 0
ret, thresh1 = cv2.threshold(imageGray, 100, 255, cv2.THRESH_TRUNC) # Truncation thresholding, if pixel value is over 100, then set it to 255
ret2, thresh2 = cv2.threshold(imageGray, 100, 255, cv2.THRESH_TOZERO) # TO Zero thresholding, if pixel value is under 100, then set it to 0
cv2.imshow("Original", imageResize) #showing original image for check
cv2.imshow("Grayscale", imageGray) # showing grayscale image
cv2.imshow("Binary", thresh) # showing results of Binary threshold
cv2.imshow("Truncation", thresh1) # showing results of Truncation threshold
cv2.imshow("To Zero", thresh2) # showing results of To Zero threshold

cv2.waitKey(0)
```

My output for original image (**cv2.imshow("Original", imageResize)**):



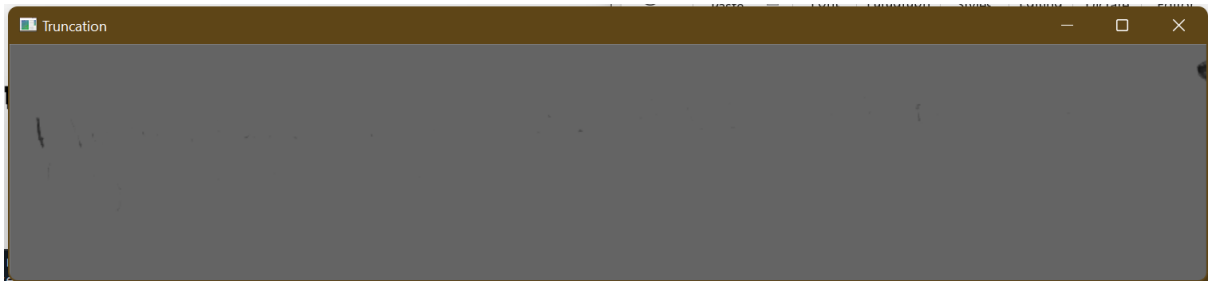
My output for grayscale image (**cv2.imshow("Grayscale", imageGray)**):



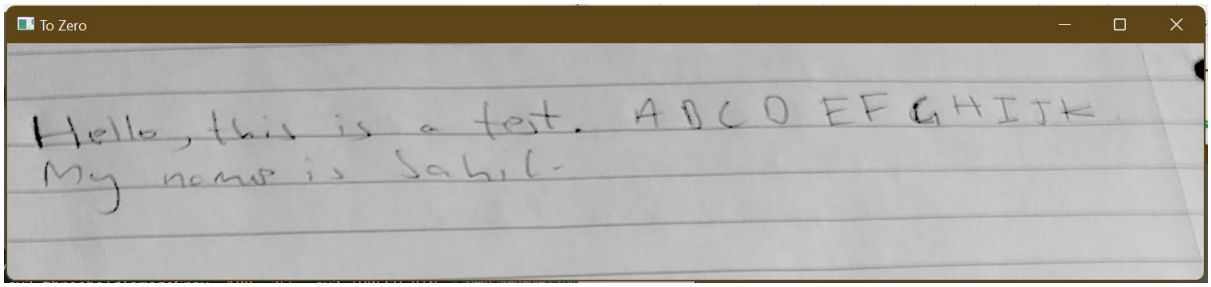
My output for Binary threshold image (**cv2.imshow("Binary", thresh)**):

My output for Truncation threshold image (**cv2.imshow("Truncation", thresh1)**):

My project: Handwriting recognition research project



My output for To Zero threshold image (cv2.imshow("To Zero", thresh2)):



Comments and justification on Thresholding:

The To Zero thresholding isn't doing anything to the original image. Therefore, I do not think this thresholding algorithm is feasible for my program as it is barely doing anything. I will try and change the parameters of cv2.threshold(imageGray, 100, 255, cv2.THRESH_TOZERO) to see if it generates any valuable output. On the other hand, Truncation and Binary thresholding algorithms have done too much (removed all the foreground information from the original image). This is convincing to me because it shows that it is a powerful tool to clean and remove noise from the image (and also increasing and giving even lighting to the image (for Binary thresholding)). I will need to change the parameter 100 to a higher value (so most of the foreground of the image is intact) in cv2.threshold(imageGray, 100, 255, cv2.THRESH_BINARY) and cv2.threshold(imageGray, 100, 255, cv2.THRESH_TRUNC).

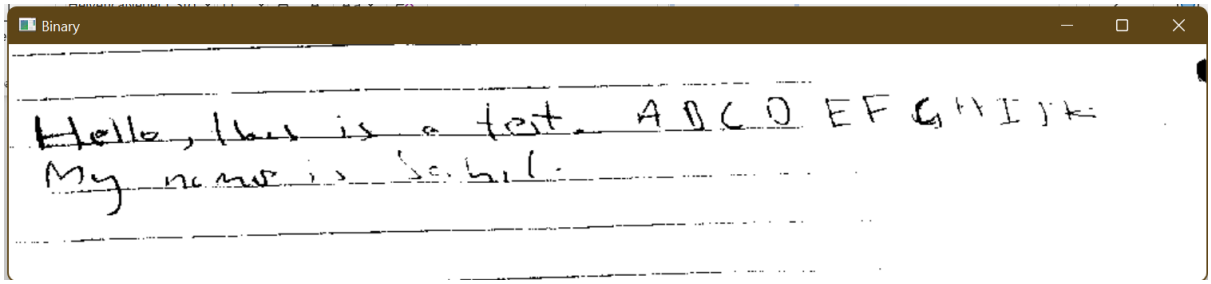
Code after changes to threshold:

```
import cv2
imageOpen = cv2.imread("input_for_doc.jpg") # read image
width = int(imageOpen.shape[1]*50/100) # width of image is reduced by scale factor 2
height = int(imageOpen.shape[0]*50/100) # height of image is reduced by scale factor 2
size1 = (width,height)
img = cv2.resize(imageOpen, size1) # cv2 method resize() resizes the original image
imageResize = cv2.resize(imageOpen, size1)
imageGray = cv2.cvtColor(imageResize, cv2.COLOR_BGR2GRAY) # converting image to grayscale as thresholding only accepts grayscale images
ret, thresh = cv2.threshold(imageGray, 150, 255, cv2.THRESH_BINARY) # Binary thresholding, if pixel value is over 100, then set it to 255, otherwise 0
ret, thresh1 = cv2.threshold(imageGray, 150, 255, cv2.THRESH_TRUNC) # Truncation thresholding, if pixel value is over 100, then set it to 255
ret2, thresh2 = cv2.threshold(imageGray, 150, 255, cv2.THRESH_TOZERO) # TO Zero thresholding, if pixel value is under 100, then set it to 0
cv2.imshow("Original", imageResize) #showing original image for check
cv2.imshow("Grayscale", imageGray) # showing grayscale image
cv2.imshow("Binary", thresh) # showing results of Binary threshold
cv2.imshow("Truncation", thresh1) # showing results of Truncation threshold
cv2.imshow("To Zero", thresh2) # showing results of To Zero threshold

cv2.waitKey(0)
```

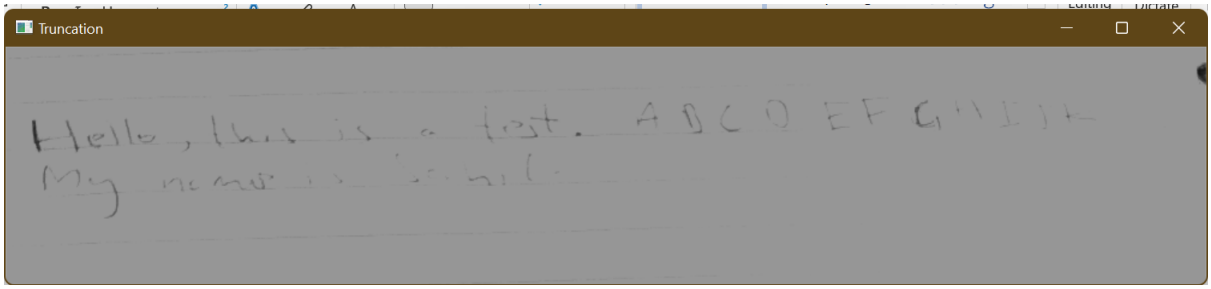
Increased threshold parameter for all 3 algorithms

My project: Handwriting recognition research project

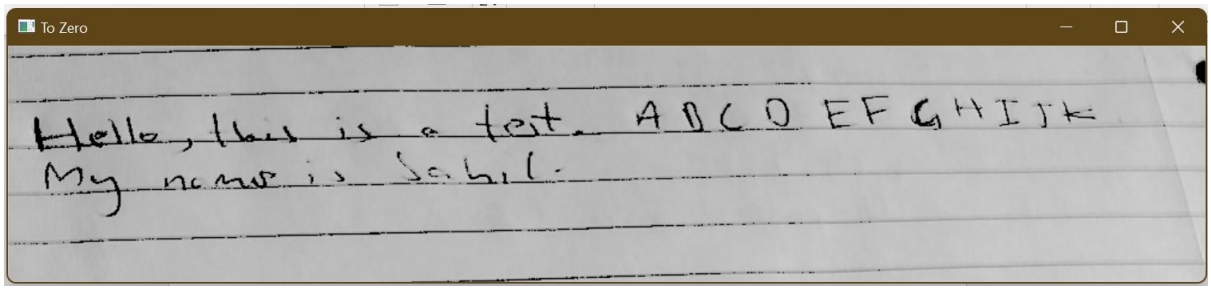
My output for Binary threshold image (cv2.imshow("Binary", thresh)):



My output for Truncation threshold image (cv2.imshow("Truncation", thresh1)):



My output for To Zero threshold image (cv2.imshow("To Zero", thresh2)):



Comments and justification on Thresholding:

I am impressed with Binary and To Zero thresholding because both of them highlights the foreground (the handwritten letters) over the background (the lined paper lines). On the other hand, Truncation thresholding outputs a faint and dull image where the foreground can be barely seen. I am more inclined to use Binary thresholding (as I said in the Design section 2.2.2) because it mostly eliminates background noise (i.e. removes the lined paper lines) and highlights the handwritten letters (also while brightening up the image) whereas, while To Zero thresholding highlights the handwritten letters, it doesn't do much to eliminate background noise and the output picture looks overall dull as compared to binary threshold, which may cause problems for letter recognition later on.

My code with just Binary thresholding:

My project: Handwriting recognition research project

```python
import cv2
imageOpen = cv2.imread("input_for_doc.jpg") # read image
width = int(imageOpen.shape[1]*50/100) # width of image is reduced by scale factor 2
height = int(imageOpen.shape[0]*50/100) # height of image is reduced by scale factor 2
size1 = (width,height)
imageResize = cv2.resize(imageOpen, size1) # cv2 method resize() resizes the original image
imageResize = cv2.blur(imageResize, (2, 2)) # blurs the image to try and reduce the background noise
imageGray = cv2.cvtColor(imageResize, cv2.COLOR_BGR2GRAY) # converting image to grayscale as thresholding only accepts grayscale images
ret, thresh = cv2.threshold(imageGray, 150, 255, cv2.THRESH_BINARY) # Binary thresholding, if pixel value is over 100, then set it to 255, otherwise 0
cv2.imshow("Original", imageResize) #showing original image for check
cv2.imshow("Grayscale", imageGray) # showing grayscale image
cv2.imshow("Binary", thresh) # showing results of binary threshold
cv2.imwrite("denoised_image.jpg", thresh)
cv2.waitKey(0)
```

I save the new thresholding image as "denoised_image.jpg" in the local directory

I added the **blur()** method which blurs the original image and minimizes noise

Test	Input data/input code	Expected result	Actual result	Pass/Fail	Explanation/Justification
Check if denoising function works, and reduces noise in input images	cv2.imread("image2.jpg")	Observational test, check for clear and noise free output image	Before:  After: 	Pa	The thresholding image has added some noise in the existing image, but it has also highlighted the handwritten text well (in my opinion) so it passed my test. I expected this test to pass because I have used thresholding a lot in the past and I know the huge benefits of using cv2 and cv2.threshold()

Section 3.3.2) Detecting individual letters

For letter detection and saving each letter as an individual image, I referred to this StackOverflow post [python - How to save OpenCV image with contour - Stack Overflow] where one of the post talks about using **cv2.BoundingRect()** to record contours and then save it using NumPy slicing in relation to the original image.

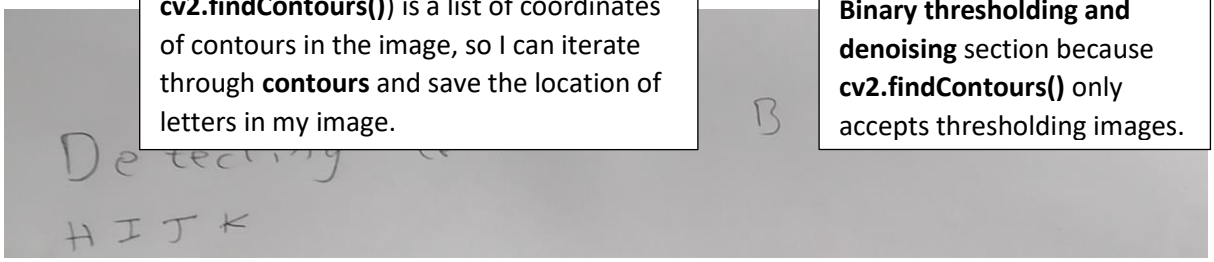
This is my code:

```
import cv2
import numpy as np
imageOpen = cv2.imread("detecting_image_test.jpg") # read image
width = int(imageOpen.shape[1]*50/100) # width of image is reduced by scale factor 2
height = int(imageOpen.shape[0]*50/100) # height of image is reduced by scale factor 2
size1 = (width,height)
imageResize = cv2.resize(imageOpen, size1) # cv2 method resize() resizes the original image
imageResize = cv2.blur(imageResize, (2, 2)) # blurs the image to try and reduce the background noise
imageGray = cv2.cvtColor(imageResize, cv2.COLOR_BGR2GRAY) # converting image to grayscale as thresholding only accepts grayscale images
ret, thresh = cv2.threshold(imageGray, 145, 255, cv2.THRESH_BINARY) # Binary thresholding, if pixel value is over 100, then set it to 255, otherwise 0
contours, h = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) # this is the method of cv2 that finds contours in an image (i.e. edges)
# the array contours contains coordinates (x,y) of each contour detected (where it starts and ends) in the image
for c in range(0, len(contours)): # I can use contour coordinates to locate letters, I will therefore use a loop to print each contour detected in the image
    decreasing_fac = 2
    x, y, w, h = cv2.boundingRect(contours[c]) # this cv2.boundingRect() outputs the x,y coordinates and the width and height of each contour, I can use this information to plot rectangles on the image where contours are.
    area_of_contour = cv2.contourArea(contours[c]) # cv2.contourArea outputs the area of the detected contour, I could have made this function myself but this method is more mathematical and gives an accurate measure of area.
    if round(area_of_contour) < 50.0: # if the area of contour is less than a certain threshold, I will count the counter as noise and therefore ignore it (next iteration of contours)
        continue
    rect = cv2.rectangle(imageResize, (x, y), (x + w, y + h), (0, 255, 0)) # plotting a rectangle where the contour is, using outputs of cv2.BoundingRect()
cv2.imshow("contours", imageResize)
cv2.waitKey(1)
```

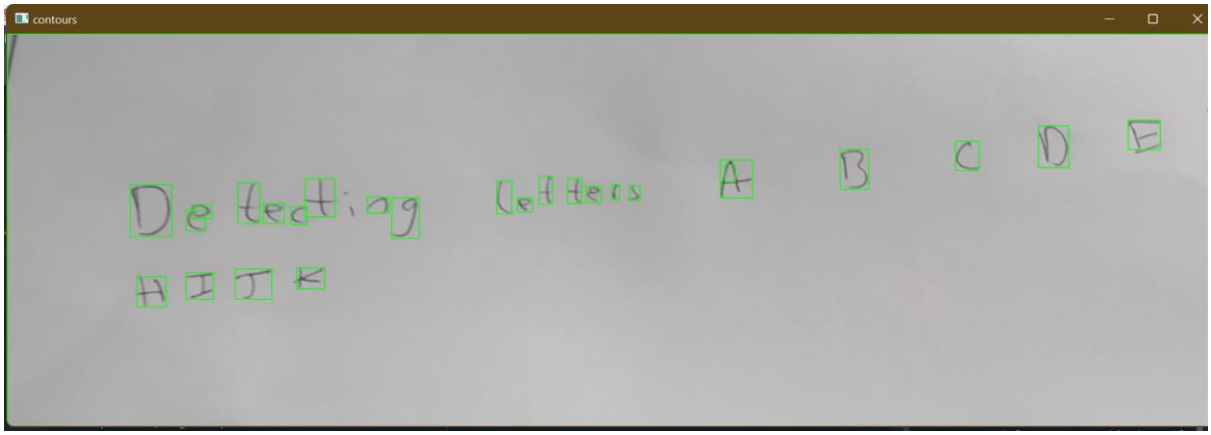
I found out that **contours** (outputs of **cv2.findContours()**) is a list of coordinates of contours in the image, so I can iterate through **contours** and save the location of letters in my image.

I reused the code from my **Binary thresholding and denoising** section because **cv2.findContours()** only accepts thresholding images.

I tested th…age_test…



The output of my program:

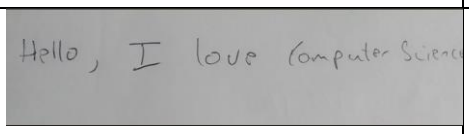
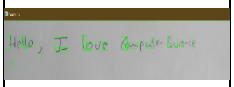
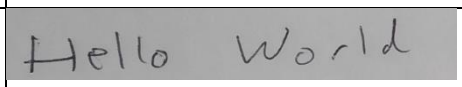
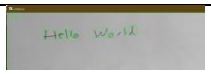


My program is successful in detecting letters in the image. In addition, it did not detect any noise as letters which is very convincing to me, meaning no false letter information can be feed into my neural network.

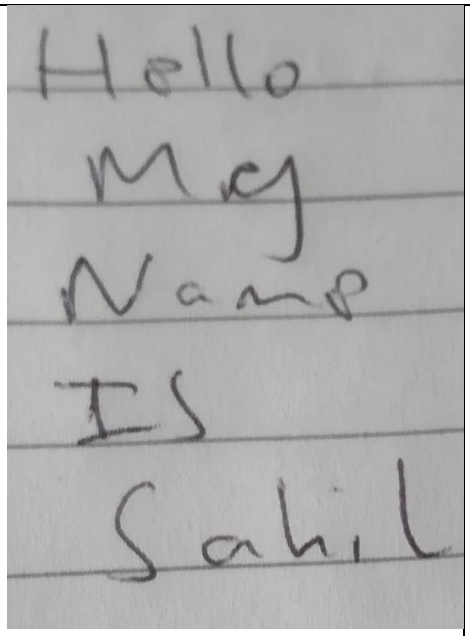
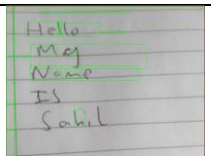
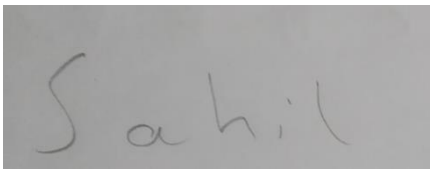
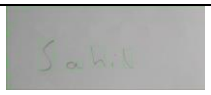
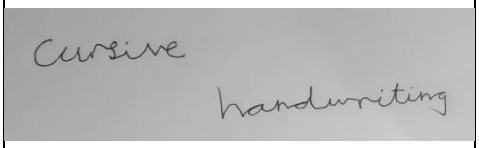
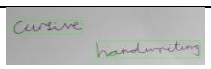
I will now use Testing data (first point on Section 2.7.2) for reference to test my algorithm

Test	Input data/input code	Desired output	Actual output	Pass	Explanation/Justification

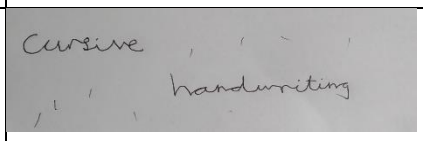
My project: Handwriting recognition research project

				s/Fail	
I will input an image with no noise and a plain white background		I expect each of the letters to be detected.		Pass	My program detected most of the words, with which I am pleased with. I think since I downsized the image during my program, the image might have lost some information (due to shrinking) hence why some letters not being detected.
I will input an image with sufficient spacing between every letter in every word		Since there is no noise in the image, I only expect the letters to be detected		Pass	This time, my program successfully detected all of my words which is very convincing. However, it has also detected some noise as letters which is a bit erroneous.

My project: Handwriting recognition research project

In this input image I will use lined paper instead of a plain white background to see if it produces erroneous detection output		I want only the letters in the image to be detected, however I think that the lines on lined paper will also be detected (which I don't want)		Fail	In my desired output, I wanted the program to detect letters, however the program acts erroneous and detects a lot of noise and not any letters. I think the there is too much noise in this image for letters to be detected consistantly
I will use faint pen/pencil writing for this image to see if letter detection still happens or not		I expect the letters to be detected because the text is too faint		Pass	Once again the program detected the letters in the image even though they are faint.
I will use cursive writing here because I want to see how my program processes this		Since the letters are all joint up, I do not expect individual letters to be detected but		Pass	As I expected, since the letters are all joint up and not individual, the letters will not be detected individually. Therefore,

My project: Handwriting recognition research project

		the whole word to be detected.			my interface cannot accept cursive handwriting.
I will input a noisy image here to check how my program handles it (i.e. does it detect noise as letters or not)		Since there is a lot of noise in the image which is hard to eliminate, I expect the noise to be detected as well as the words		Pass	This matches my predictions that the unnecessary edges (noise) in this image was recognised as letters because it was too thick and cannot be eliminated by blurring or thresholding.

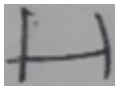
Before I finish with my letter detection program, I need a way to save all my detected letters in image format so this is my code for it:

```
import cv2
import numpy as np
imageOpen = cv2.imread("faint_pencil_for_detection.jpg") # read image
width = int(imageOpen.shape[1]*30/100) # width of image is reduced by scale factor 2
height = int(imageOpen.shape[0]*30/100) # height of image is reduced by scale factor 2
size1 = (width,height)
imageResize = cv2.resize(imageOpen, size1) # cv2 method resize() resizes the original image
imageResize = cv2.blur(imageResize, (2, 2)) # blurs the image to try and reduce the background noise
imageGray = cv2.cvtColor(imageResize, cv2.COLOR_BGR2GRAY) # converting image to grayscale as thresholding only accepts grayscale ima
ret, thresh = cv2.threshold(imageGray, 150, 255, cv2.THRESH_BINARY) # Binary thresholding, if pixel value is over 100, then set it t
contours, h = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) # this is the method of cv2 that finds contours in ar
# the array contours contains coordinates (x,y) of each contour detected (where it starts and ends) in the image
num = 0
for c in range(0, len(contours)): # I can use contour coordinates to locate letters, I will therefore use a loop to print each conto
    x, y, w, h = cv2.boundingRect(contours[c]) # this cv2.boundingRect() outputs the x,y coordinates and the width and height of eac
    area_of_contour = cv2.contourArea(contours[c]) # cv2.contourArea outputs the area of the detected contour, I could have made thi
    if round(area_of_contour) < 50.0: # if the area of contour is less than a certain threshold, I will count the counter as noise a
        continue
    num += 1
    crop = imageResize[y:y+h, x:x+w] # I use numpy slicing to extract the region of the image containing the contour
    name = "letter_image{}.jpg".format(num)
    cv2.imwrite(name, crop) # saving each contour
    rect = cv2.rectangle(imageResize, (x, y), (x + w, y + h), (0, 255, 0)) # plotting a rectangle where the contour is, using output
cv2.imshow("contours", imageResize)
cv2.waitKey(0)
```

I used NumPy slicing here to extract the portion of the original image that corresponds to the contour and save it as a .jpg image

I will save this program as **"detect_letters.py"** and I will keep referring to this code, because I will also require this later on in my development when I need corresponding coordinates to each letter in the image for sorting out words and forming sentences with machine learning output.

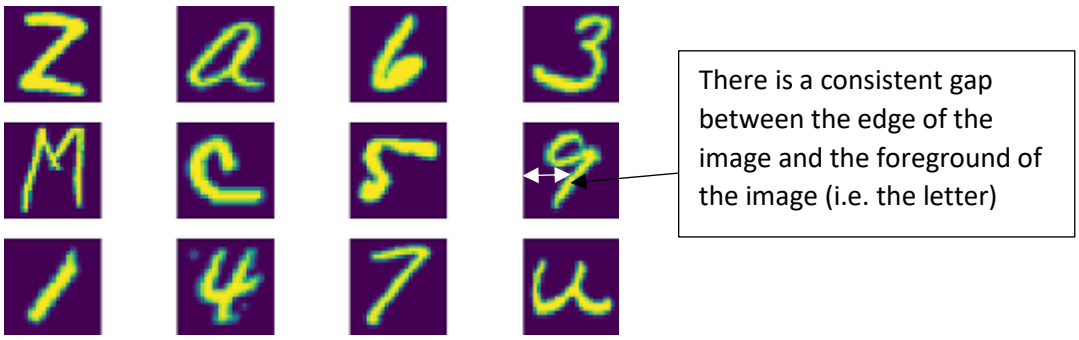
The saved images (**"letter_image[number].jpg"**) look like this:



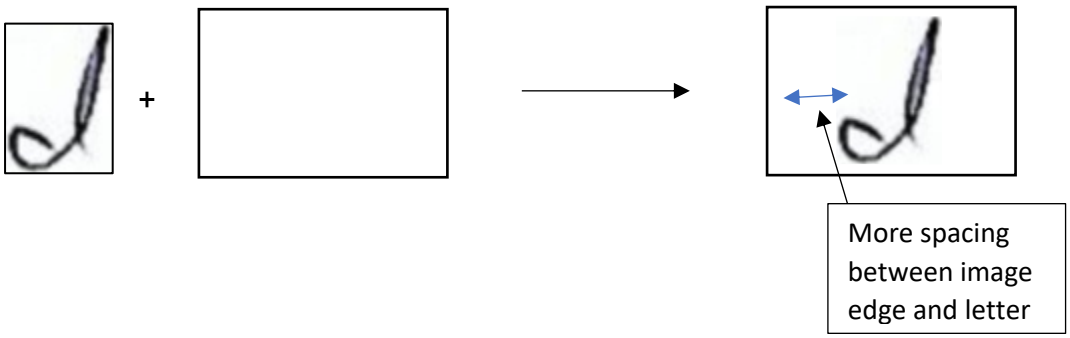
Section 3.3.3) Making spacing for letters in letter images

As you can see from the images above, the letter in the image is too close to the edge of image frame in each case, which concerns me. I think that this can create anomalies when these images will be fed into my neural network because the image format doesn't match the EMNIST images. As you can see below, EMNIST images have sufficient spaces between the edge of the image and the contours of the letter. I must write an algorithm that can take my existing letter image and put it the centre of a bigger (blank) image so that the letter has sufficient space from the edge of the image.

EMNIST images for reference above:



My plan for spacing between image and letter:



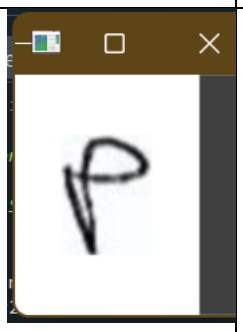
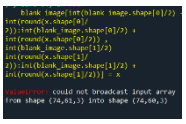
My project: Handwriting recognition research project

My code for this:

```
import numpy as np
import cv2
def main(x): # since this is meant to be used for multiple images, I will make a subroutine instead of a sequence, so multiple the function can be called several times for use instead of repeating a sequence (unnessary lines of code)
    width = x.shape[1] # since x is cv2.imread() array, I can use .shape() method to retrieve width and height of the image
    height = x.shape[0]
    blank_image = np.full((height*2, width*2, 3), 255, dtype = np.uint8) # np.full() creates a numpy array with (height*2, width*2, 3) , 255 , dtype = np.uint8) is the dimensions of a multiplied by a scale factor 2
    blank_image[int(blank_image.shape[0]/2) - int(round(x.shape[0]/2)):int(blank_image.shape[0]/2) + int(round(x.shape[0]/2)) , int(blank_image.shape[1]/2) - int(round(x.shape[1]/2)):int(blank_image.shape[1]/2) + int(round(x.shape[1]/2))] = x
    # since we need x to be in the centre of blank_image, we need to substitute a section of blank_image(x coordinate) , half of x(its x coordinate), and y centre of blank_image(y coordinate) - half of x (its x coordinate), and y = centre of blank_image(y coordinate) + half of x(its y coordinate) and centre of blank_image(y coordinate) - half of x (its y coor
    cv2.imshow("blank_image", blank_image)
a = cv2.imread("Letter_image10.jpg")
main(a)
cv2.waitKey(0)
```

This is the main bit of the program where the centre of the **blank_image** is assigned to **x** by using some basic NumPy slicing knowledge.

Test on "spacing_for_letter.py"

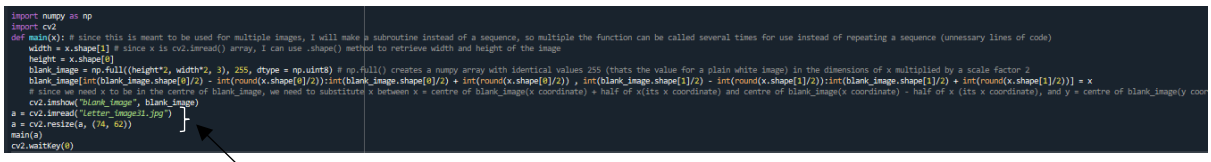
Test	Input data/code	Desired output	Actual output	Pass/Fail	Explanation/Justification
I will test this program on one of the letter images outputted in "detect_letters.py"		The input image should be at the centre of another image (double the size of input) and there should be spacing between the edge of the image and the letter.		Pass	The input image is at the centre of the second (larger) blank image, and there is a consistent gap between the end of the image and the letter. Therefore, this passes my test.
I will test this program on one of the letter images outputted in		The input image should be at the		Fail	This is a new error for me. The dimensions of the input image are odd (74, 61) meaning when I do NumPy

"detect_letters.py"		centre of another image (double the size of input) and there should be spacing between the edge of the image and the letter.			slicing, one of my values inside the slice will be a decimal (NumPy slicing doesn't work with decimal parameters). However, I have used the round() so I will not get a decimal parameter. I will investigate this problem further.

Fixing my test:

To work out where my program has gone wrong, I must first understand why the program gave the right output for the first input. The first input had dimensions (60, 46) which are both even. That is the only difference between the format of my first input and second input.

To diagnose the problem, I will alter the dimensions of my second input from (74, 61) to (74, 62) which is even and see if my program works.



I used **cv2.resize()** to change dimensions

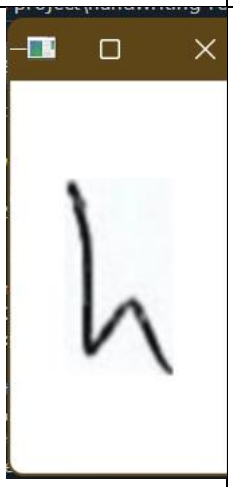
The output:

My project: Handwriting recognition research project

The program works with even dimensions! Therefore, I need to amend my program, to check for dimensions and change them if they are odd.

My code:

Test on "spacing_for_letter.py" (attempt 2)

Test	Input data/code	Desired output	Actual output	Pass/Fail	Explanation/Justification
I will test this program on one of the letter images outputted in "detect_letters.py" (with odd dimensions)	Dimensions (97, 53) 	The input image should be at the centre of another image (double the size of input) and there should be spacing between the edge of the image and the letter.		Pass	The input image is at the centre of the second (larger) blank image, and there is a consistent gap between the end of the image and the letter. Even though it has odd dimensions the program corrected it and outputted the new image. Therefore, this passes my test.

This is my final code for "spacing_for_letter.py":

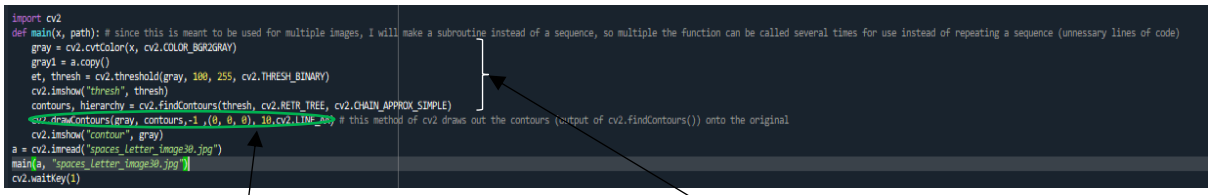
My project: Handwriting recognition research project

I just added cv2.imwrite() so I can save the new letter image.

Section 3.3.4) Making a thickness enlargement code for letters

Now that I have made a program that can space out letters inside its individual image, I need to make the next program that was in my Key features for solution (point 2 of Section 2.7.2) which is thickness enlargement of each letter (after using "spacing_for_letter.py" on it). The python file will be called "thickness_of_letter_enlarged.py".

This is my code for "thickness_of_letter_enlarged.py":



```
import cv2
def main(x, path): # since this is meant to be used for multiple images, I will make a subroutine instead of a sequence, so multiple the function can be called several times for use instead of repeating a sequence (unnessary lines of code)
    gray = cv2.cvtColor(x, cv2.COLOR_BGR2GRAY)
    gray1 = a.copy()
    et, thresh = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
    cv2.imshow("thresh", thresh)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(gray, contours,-1 ,(0, 0, 0), 10,cv2.LINE_AA) # this method of cv2 draws out the contours (output of cv2.findContours()) onto the original
    cv2.imshow("contour", gray)
a = cv2.imread("spaces_letter_image30.jpg")
main(a, "spaces_letter_image30.jpg")
cv2.waitKey(1)
```

cv2.drawContours() is what is used to draw and highlight contours using coordinates from **cv2.findContours()**. (0,0,0) (black)is the colour of the highlighting, and 10 is the thickness of the highlighted lines.

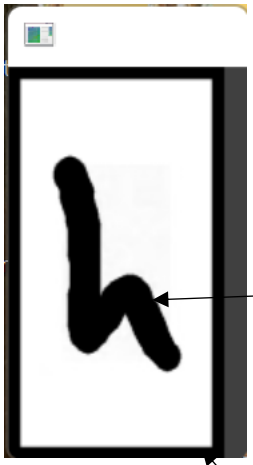
I have copied this section of the code from my **"detect_letters.py"** program because to enlarge my contours in the image, I will need to find the contours in the image first, and that involves the same steps as **"detect_letters.py"**

Test run:

My input image is this (its also the output of "spacing_for_letter.py"):

The output of "thickness_of_letter_enlarged.py" is:

My project: Handwriting recognition research project



The program prints out a thicker outline of my letter which is what I wanted.

I need to write code to get rid of this thick black border that has formed in the output.

My amended code:

```
import cv2
def main(x, path): # since this is meant to be used for multiple images, I will make a subroutine instead of a sequence, so multiple the function can be called several times for use instead of repeating a sequence (unnessary lines of code)
    gray = cv2.cvtColor(x, cv2.COLOR_BGR2GRAY)
    gray1 = a.copy()
    et, thresh = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
    cv2.imshow("thresh", thresh)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(gray, contours,-1 ,(0, 0, 0), 10,cv2.LINE_AA) # this method of cv2 draws out the contours (output of cv2.findContours()) onto the original
    cv2.rectangle(gray,(0,0),(gray1.shape[1], gray1.shape[0]), (255,255,255),13)
    cv2.imshow("contours", gray)
a = cv2.imread("spaces_letter_image30.jpg")
main(a, "spaces_letter_image30.jpg")
cv2.waitKey(1)
```

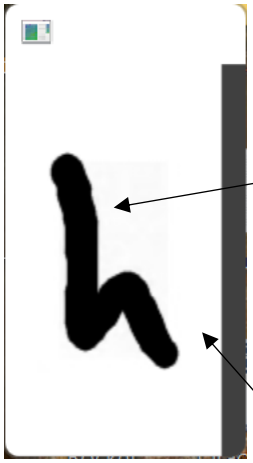
I added a rectangle that covers the whole image and has a thick white (255, 255, 255) outline to counter the black outline that formed

Test run (attempt 2):

My input image is this:

The output of **“thickness_of_letter_enlarged.py”** is:

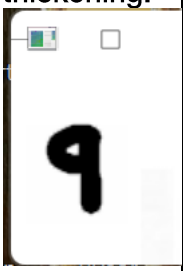
My project: Handwriting recognition research project



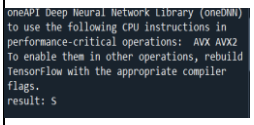
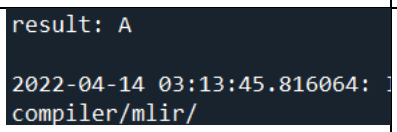
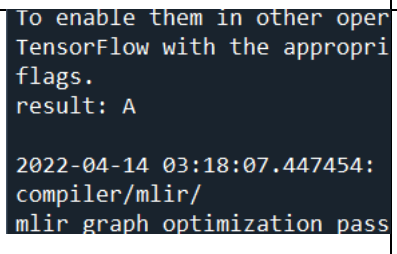
The program prints out a thicker outline of my letter which is what I wanted.

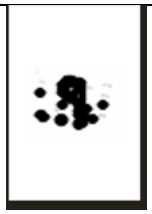
There is no black outline now, so my program works as planned.

I will now use testing data (second point of Section 2.7.2) for reference to test my thickness algorithm. In this test, I will feed the thickened contours into my neural network to see if they are recognised by the neural network

Test	Input data/Input code	Desired output	Actual output	Pass/ Fail	Explanation/Justification
I will input a letter image with no noise and only the letter to see if my neural network recognises the image correctly	Before thickening:  After thickening: 	The neural network correctly recognises the image as the digit 9	(registered 2) result: Q In [30]:	Fail	The neural network didn't recognise the digit correctly (it said the image represents "Q" which is false). I can only account this to the fact that the thickness of the letter might be too high. I can try and reduce the thickness for my next test.

My project: Handwriting recognition research project

I will input a letter image with only one letter (again to see if my neural network recognises the image correctly)	Before thickening:  After thickening (reduced thickness, as mentioned above): 	The neural network correctly recognises the image as the letter S		Pass	The neural network works well with fainter letter lines than the first test because it managed to recognise my input when I reduced the thickness factor.
To verify whether the neural network outputs the correct result if part of letter is cut during automated cropping.	I will simulate a faulty image:  After thickening: 	The neural network correctly recognises the image as the digit 4		Fail	I expected this test to fail because I do not expect the neural network to give a correct output if the input is faulty (i.e., partly cut).
To verify whether the neural network outputs the correct result	Before thickening:  After thickening:	The neural network correctly recognises the image		Fail	I expected this failed test because there is just too much noise in the image (which was amplified by my thickness enlargement program) and

if there is noise in the input image.		as the digit 9			there is no way my neural network can deduce any useful information for a reliable output if there is so much noise

After testing, this is my final code for **"thickness_of_letter_enlarged.py":**

```
import cv2
def main(x, path): # since this is meant to be used for multiple images, I will make a subroutine instead of a sequence, so multiple the function can be called several times for use instead of repeating a sequence (unnessary lines of code)
    gray = cv2.cvtColor(x, cv2.COLOR_BGR2GRAY)
    gray1 = a.copy()
    et, thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_OTSU)
    cv2.imshow("thresh", thresh)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(gray, contours,-1 ,(0, 0, 0), 2,cv2.LINE_AA) # this method of cv2 draws out the contours (output of cv2.findContours()) onto the original
    cv2.rectangle(gray,(0,0),(gray1.shape[1], gray1.shape[0]), (255,255,255),13)
    cv2.imshow("contour", gray)
    cv2.imwrite("thick_{}.png".format(path), gray)
a = cv2.imread("test_1.png")
main(a, "test_1.png")
cv2.waitKey(1)
```

During testing I found out that **thickness = 2** was a good parameter for **cv2.drawContours()** for the neural network input, so I will implement permanently in this program.

I just added cv2.imwrite() so I can save the new letter image.

Section 3.3.5) Making a GUI for testing

Now that I have made coded algorithms for 3 of my objectives in Design section 2.2.2 (image pre-processing, detecting individual letters, and making letter contours thicker) I will move on to my 4th and 5th points which are partition letters (from input image) into lines/rows (after using neural network) and grouping letters to form words (using letter positions in the image and coordinates)

Before starting to code my solutions, I wanted to make a simple graphical interface (preferably in Tkinter) where I can display my letters detected on input images using bounding boxes, while also incorporating a zoom function. This is part of my graphical user interface solution in section 2.2.3, but I need it specifically now for visually observing coordinates of letters, average gaps between words/each letter/each row, and outputting recognised neural network output on the image to measure where the recognition is accurate and where it stumbles. Of course, I can do that with just OpenCV as well, however, OpenCV is very static in the fact that it doesn't allow objects such as bounding boxes to be modular and individual (instead it is made to become part of the image its placed on, and cannot be removed or resized later on). However, Tkinter deals with each object as separate, which allows me to experiment with various aspects of image processing such as zooming (where I can resize each bounding box in relation to the changing image size).

My project: Handwriting recognition research project

My python file for Tkinter is called **testGUI.py**

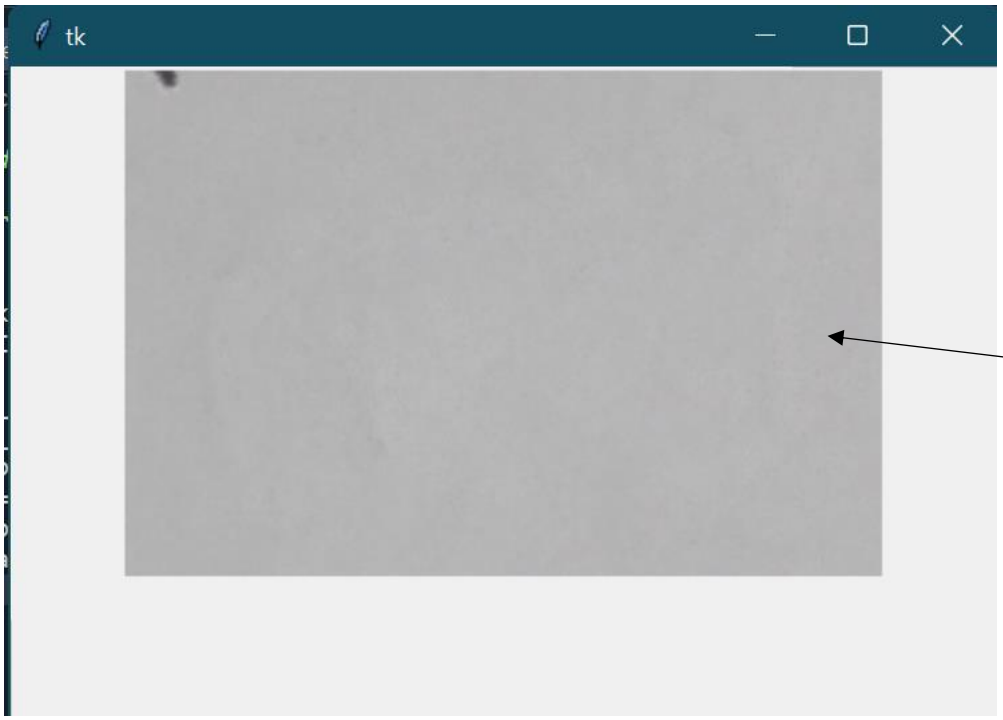
This is my code for showing my input image in a Tkinter window (in a canvas widget):

```
from tkinter import *
from PIL import Image, ImageTk
root = Tk() #defining the main window of the program
canvas_image = Canvas(root) # canvas widget is defined where I will place my input image
canvas_image.pack() # .pack() allocates a coordinate space for canvas_image
imageOpen = Image.open("plain_background_detection.jpg") # opening input image
image = ImageTk.PhotoImage(imageOpen) # PhotoImage object that will be used to output image in canvas_image
image_plot = canvas_image.create_image(0,0,image=image) # .create_image() displayes my input image on the canvas
root.mainloop()
```

Tkinter **Canvas()** is a widget where you can draw on. This gives me flexibility to draw rectangles or circles around detected letters which are resizable and modular (completely independent of the image)

I used the PIL library again because it has **ImageTk** which allows images to be shown on a Tkinter window

My output:



My input image is output in the canvas widget

As you can see I have successfully displayed my input image on a canvas widget in a Tkinter window, however, most of the image is cut and there is no way of navigating the image in any way. To solve this problem, I must implement scrollbars (horizontal and vertical) that allow me to navigate the image.

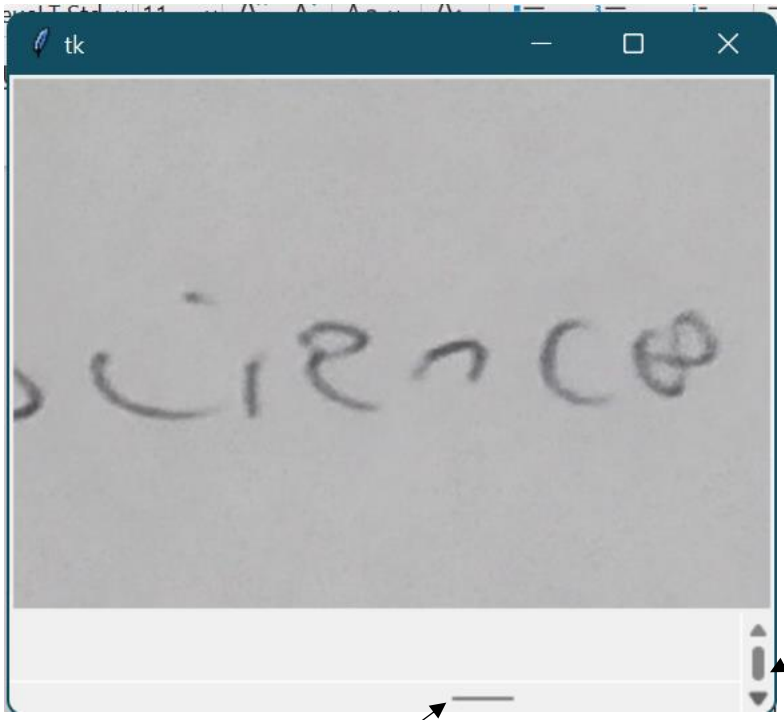
This is my code for scrollbars:

My project: Handwriting recognition research project

```
from tkinter import *
from PIL import Image, ImageTk
root = Tk() #defining the main window of the program
canvas_image = Canvas(root) # canvas widget is defined where I will place my input image
canvas_image.pack() # .pack() allocates a coordinate space for canvas_image
imageOpen = Image.open("plain_background_detection.jpg") # opening input image
image = ImageTk.PhotoImage(imageOpen) # PhotoImage object that will be used to output image in canvas_image
image_plot = canvas_image.create_image(0,0,image=image) # .create_image() displayes my input image on the canvas
vertical_scrollbar = Scrollbar(root, orient = "vertical", command = canvas_image.yview) # defining vertical scrollbar
vertical_scrollbar.pack(side = RIGHT, fill = Y)
horizontal_scrollbar = Scrollbar(root, orient = "horizontal", command = canvas_image.xview) # defining horizontal scrollbar
horizontal_scrollbar.pack(side = BOTTOM, fill = X) # placement of this scrollbar is at the bottom
canvas_image.configure(yscrollcommand = vertical_scrollbar.set, xscrollcommand = horizontal_scrollbar.set) # this is where I assign the scrollbar command onto canvas_image specifically
canvas_image.configure(scrollregion=canvas_image.bbox("all"))
root.mainloop()
```

This is my code for scrollbars (vertical and horizontal). Since I was not sure about how to implement scrollbars, I referred to this link [python - Tkinter, make scrollbar stick to canvas and not root? - Stack Overflow]

After executing this code, my Tkinter window is below:



My vertical scrollbar is tiny as compared to the window, but it works and elegantly scrolls my input image up and down. Since, this interface is just for testing, I will not bother with remapping this scrollbar to make it more convenient to use.

My horizontal scrollbar works perfectly as well

Before I start to put bounding boxes on **canvas_image**, I need to make a zoom widget so I can zoom in/out to enlarge and shrink images. To do this I will use the **Scale()**

method in Tkinter, and using the scale factor that the user sets (by sliding the image), I will update the image dimensions.

This is my code for zooming in/out for the input image:

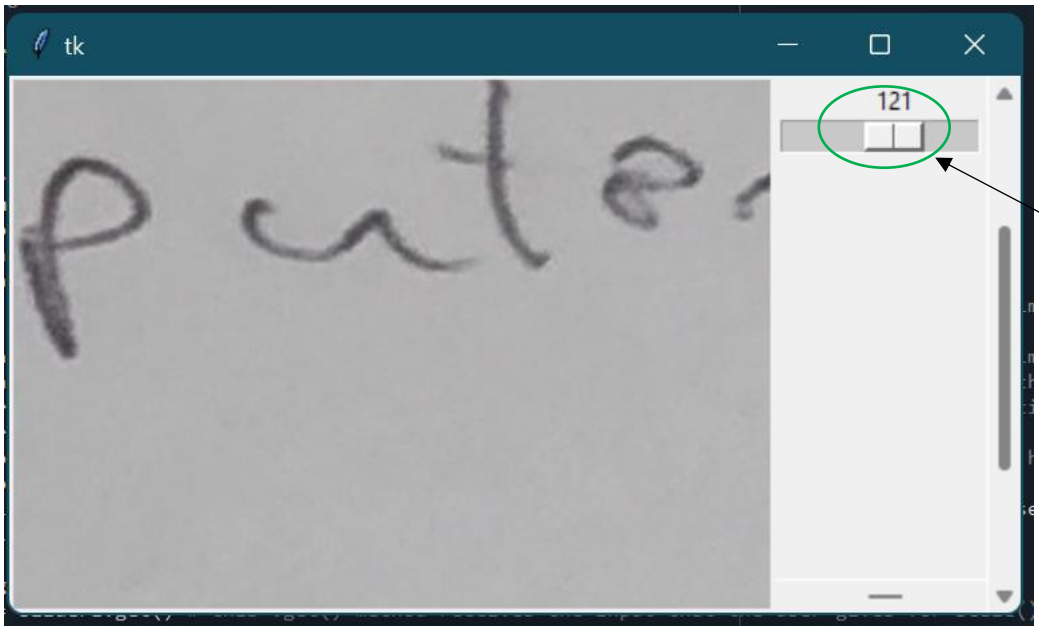
```
from tkinter import *
from PIL import Image, ImageTk
import numpy as np
root = Tk() #defining the main window of the program
canvas_image = Canvas(root) # canvas widget is defined where I will place my input image
canvas_image.pack(side=LEFT, expand=YES, fill=BOTH) # .pack() allocates a coordinate space for canva
imageOpen = Image.open("plain_background_detection.jpg") # opening input image
image = ImageTk.PhotoImage(imageOpen) # PhotoImage object that will be used to output image in canva
image_plot = canvas_image.create_image(0,0,image=image) # .create_image() displayes my input image on the canvas
vertical_scrollbar = Scrollbar(root, orient = "vertical", comman
vertical_scrollbar.pack(side = RIGHT, fill = Y)
horizontal_scrollbar = Scrollbar(root, orient = "horizontal", co
horizontal_scrollbar.pack(side = BOTTOM, fill = X) # placement o
canvas_image.configure(yscrollcommand = vertical_scrollbar.set,
canvas_image.configure(scrollregion=canvas_image.bbox("all"))
def zoom(event):
    global newImage
    scale_factor = slider1.get() # this .get() method receives t
    width = int(np.shape(imageOpen)[1]*scale_factor/100) # I change the width of the original image using scale factor
    height = int(np.shape(imageOpen)[0]*scale_factor/100) # I change the height of the original image using scale factor
    imageResize = imageOpen.resize((width, height)) # resizing image based on new width and height
    newImage = ImageTk.PhotoImage(image = imageResize)
    canvas_image.itemconfig(image_plot, image = newImage) #  itemconfig is used to update imageplot which deals with displaying images (in above code)
slider1 = Scale(root, from_ = 0, to = 200, orient = HORIZONTAL, command = zoom) # this is the slider that is used for zooming, with minimum value = 0, maximum value = 200
slider1.pack()
root.mainloop()
```

I also changed the **.pack()** method for **canvas_image**, so now it is on the left of the window instead of centre.

This is where I resize the image and display it as a new canvas image

Scale() is the Tkinter method that is a sliding widget. The user can slide this widget, which will call the function **zoom()**

My output after running this code:



The slider successfully changes the dimensions of the canvas image

While testing the zoom() function, I get an error when scale_factor = 0 in zoom():

My project: Handwriting recognition research project

```
  File "c:\users\sahil\desktop\computing
project\handwriting-recognition-ann-
master\image_recog_stuff\test.py", line 28, in zoom
    imageResize = imageOpen.resize((width, height)) #
resizing image based on new width and height
  File "C:
\Users\Sahil\AppData\Local\Programs\Python\Python39\l
ib\site-packages\PIL\Image.py", line 1980, in resize
    return self._new(self.im.resize(size, resample,
box))
ValueError: height and width must be > 0
```

The **Value error** happens because the width and height of the new **imageResize** image is 0 which is not accepted by **PIL .resize()** method. To fix this, I made the following changes to my **zoom()** function:

```
def zoom(event):
    global newImage
    scale_factor = slider1.get() # this .get() method receives the input that
    width = int(np.shape(imageOpen)[1]*scale_factor/100) # I change the width
    height = int(np.shape(imageOpen)[0]*scale_factor/100) # I change the heigh
    if width == 0:
        width = 1
    if height == 0:
        height = 1
    imageResize = imageOpen.resize((width, height)) # resizing image based on
    newImage = ImageTk.PhotoImage(image = imageResize)
    canvas_image.itemconfig(image_plot, image = newImage) # .itemconfig is use
```

I check the values for **width** and **height** before being passed into **.resize().** If width or height is zero, then I assign it the smallest non-zero positive integer which is 1.

For plotting bounding boxes onto a Tkinter canvas, I must verify that OpenCV coordinates for images are interchangeable with Tkinter coordinates for images (e.g., information at point (4,5) is the same in OpenCV as it is in Tkinter Canvas).

For this I will do a small test. I will create a blank image in OpenCV, and plot a dot at a given location. I will then import the image to my Tkinter program above, and use canvas.bind() to find the coordinates of the dot when I hover my cursor over there. If the dot locations from OpenCV and Tkinter align, then it passes my test and I can use my OpenCV coordinates for Tkinter.

Code for OpenCV

```
import cv2
import numpy as np
blank_image = cv2.imread("blank_image.png") # opens an image that is blank
cv2.circle(blank_image, (34,60), 2, (0,0,0)) # draws a circle of radius 5 (that is black) at centre (34, 60)
cv2.imwrite("new_image.png", blank_image)
cv2.waitKey()
```

Image with dot (plot at (34, 60)):

My project: Handwriting recognition research project

Amendments of my Tkinter code:

```
imageOpen = Image.open("new_image.png") # opening input image
image = ImageTk.PhotoImage(imageOpen) # PhotoImage object that
```

I changed my image path to “new_image.png”

```
def getcoor(eventorigin):
    x = eventorigin.x
    y = eventorigin.y
    print("x{}, y{}".format(x,y))
canvas_image.bind('<Button 1>', getcoor)
```

canvas_image.bind() outputs coordinates of cursor, I will print these coordinates, and compare it against (34, 60)

Output to my Tkinter code:

```
x37, y63
x37, y63
x35, y62
x35, y62
x38, y63
x39, y64
```

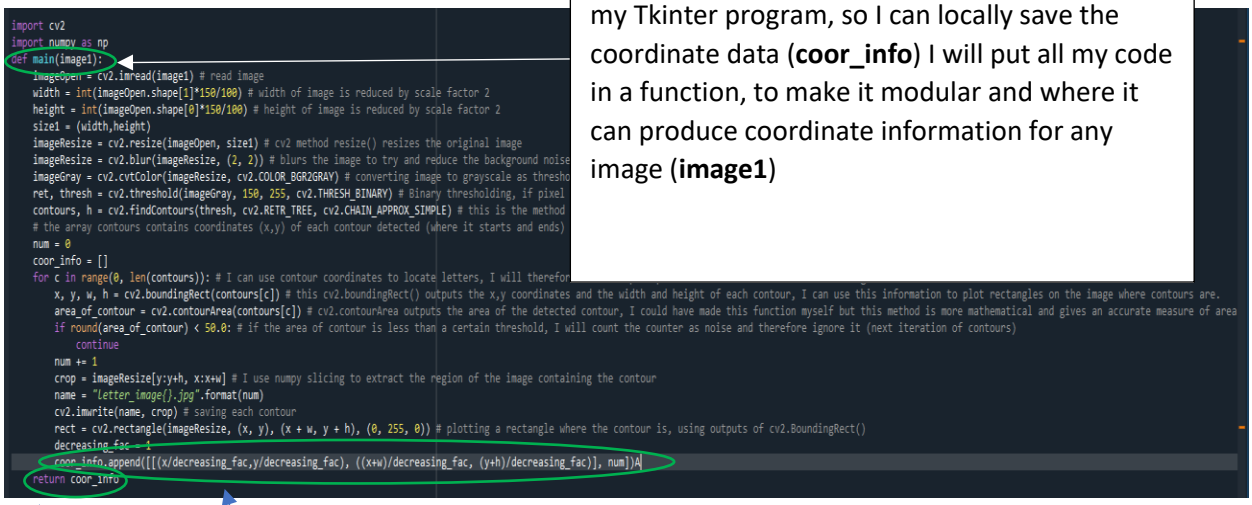
When I take my cursor to the dot, **getcoor()** function consistently outputs coordinates near (34, 60) so I am convinced that the coordinate system in Tkinter and OpenCV are the same.

Now that my test has passed for coordinate check, I will create bounding boxes for my input handwritten data.

To create my bounding boxes, I will need coordinate data from detect_letters.py because this is where I extract my individual letters from and save them as its own image path. Therefore I must amend the code in detect_letters.py to output a coordinates list, that can be used in my Tkinter program.

My project: Handwriting recognition research project

First of all, since I plan to invoke this program in my Tkinter program, so I can locally save the coordinate data (**coor_info**) I will put all my code in a function, to make it modular and where it can produce coordinate information for any image (**image1**)



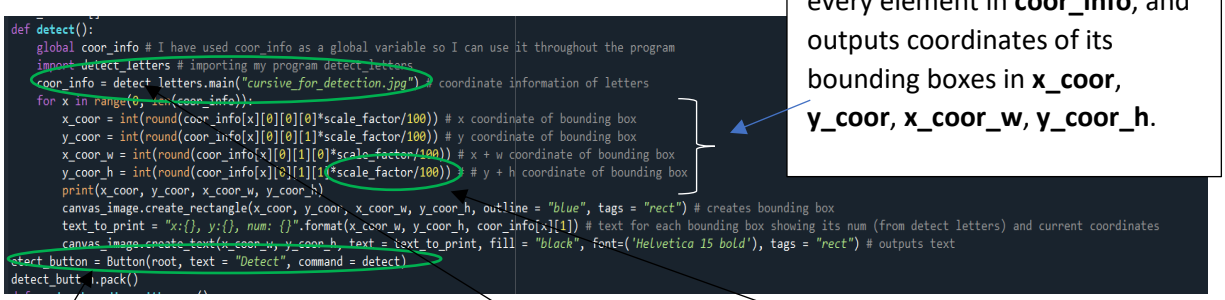
```
import cv2
import numpy as np
def main(image1):
    imageOpen = cv2.imread(image1) # read image
    width = int(imageOpen.shape[1]*150/100) # width of image is reduced by scale factor 2
    height = int(imageOpen.shape[0]*150/100) # height of image is reduced by scale factor 2
    size1 = (width,height)
    imageResize = cv2.resize(imageOpen, size1) # cv2 method resize() resizes the original image
    imageResize = cv2.blur(imageResize, (2, 2)) # blurs the image to try and reduce the background noise
    imageGray = cv2.cvtColor(imageResize, cv2.COLOR_BGR2GRAY) # converting image to grayscale as thresho
    ret, thresh = cv2.threshold(imageGray, 150, 255, cv2.THRESH_BINARY) # Binary thresholding, if pixel
    contours, h = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) # this is the method
    # the array contours contains coordinates (x,y) of each contour detected (where it starts and ends)
    num = 0
    coor_info = []
    for c in range(0, len(contours)): # I can use contour coordinates to locate letters, I will therefor
        x, y, w, h = cv2.boundingRect(contours[c]) # this cv2.boundingRect() outputs the x,y coordinates and the width and height of each contour, I can use this information to plot rectangles on the image where contours are.
        area_of_contour = cv2.contourArea(contours[c]) # cv2.contourArea outputs the area of the detected contour, I could have made this function myself but this method is more mathematical and gives an accurate measure of area
        if round(area_of_contour) < 50.0: # if the area of contour is less than a certain threshold, I will count the counter as noise and therefore ignore it (next iteration of contours)
            continue
        num += 1
        crop = imageResize[y:y+h, x:x+w] # I use numpy slicing to extract the region of the image containing the contour
        name = "letter_image{}.jpg".format(num)
        cv2.imwrite(name, crop) # saving each contour
        rect = cv2.rectangle(imageResize, (x, y), (x + w, y + h), (0, 255, 0)) # plotting a rectangle where the contour is, using outputs of cv2.BoundingRect()
        decreasing_fac = 1
        coor_info.append([[(x/decreasing_fac,y/decreasing_fac), ((x+w)/decreasing_fac, (y+h)/decreasing_fac)], num])
    return coor_info
```

coor_info is returned as a result of this function.

coor_info is where the start coordinate (x,y) and the end coordinate(x+w, y+h) is added. This is the coordinates for a bounding box for the individual letter. I have also added an additional element **num**, which is just the number that is serialised into saving the individual contour in **"letter_image{}.jpg".format(num).** I will be using this number to uniquely identify each contour when I use its coordinates. **decreasing_fac** is a variable to scale the coordinates so they represent the relative positions when the width and height of the image are original

This is how I invoke my detect_letter.py program in my Tkinter program (testGUI.py):

This for loop iterates through every element in **coor_info**, and outputs coordinates of its bounding boxes in **x_coor**, **y_coor**, **x_coor_w**, **y_coor_h**.



```
def detect():
    global coor_info # I have used coor_info as a global variable so I can use it throughout the program
    import detect_letters # importing my program detect_letters
    coor_info = detect_letters.main("cursive_for_detection.jpg") # coordinate information of letters
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0]*scale_factor/100)) # x coordinate of bounding box
        y_coor = int(round(coor_info[x][0][1]*scale_factor/100)) # y coordinate of bounding box
        x_coor_w = int(round(coor_info[x][1][0]*scale_factor/100)) # x + w coordinate of bounding box
        y_coor_h = int(round(coor_info[x][1][1]*scale_factor/100)) # y + h coordinate of bounding box
        print(x_coor, y_coor, x_coor_w, y_coor_h)
        canvas_image.create_rectangle(x_coor, y_coor, x_coor_w, y_coor_h, outline = "blue", tags = "rect") # creates bounding box
        text_to_print = "x:{}, y:{}, num: {}".format(x_coor_w, y_coor_h, coor_info[x][1]) # text for each bounding box showing its num (from detect letters) and current coordinates
        canvas_image.create_text(x_coor_w, y_coor_h, text = text_to_print, fill = "black", font=('Helvetica 15 bold'), tags = "rect") # outputs text
detect_button = Button(root, text = "Detect", command = detect)
detect_button.pack()
```

I made a button **detect_button**, if it is pressed **detect()** function is called

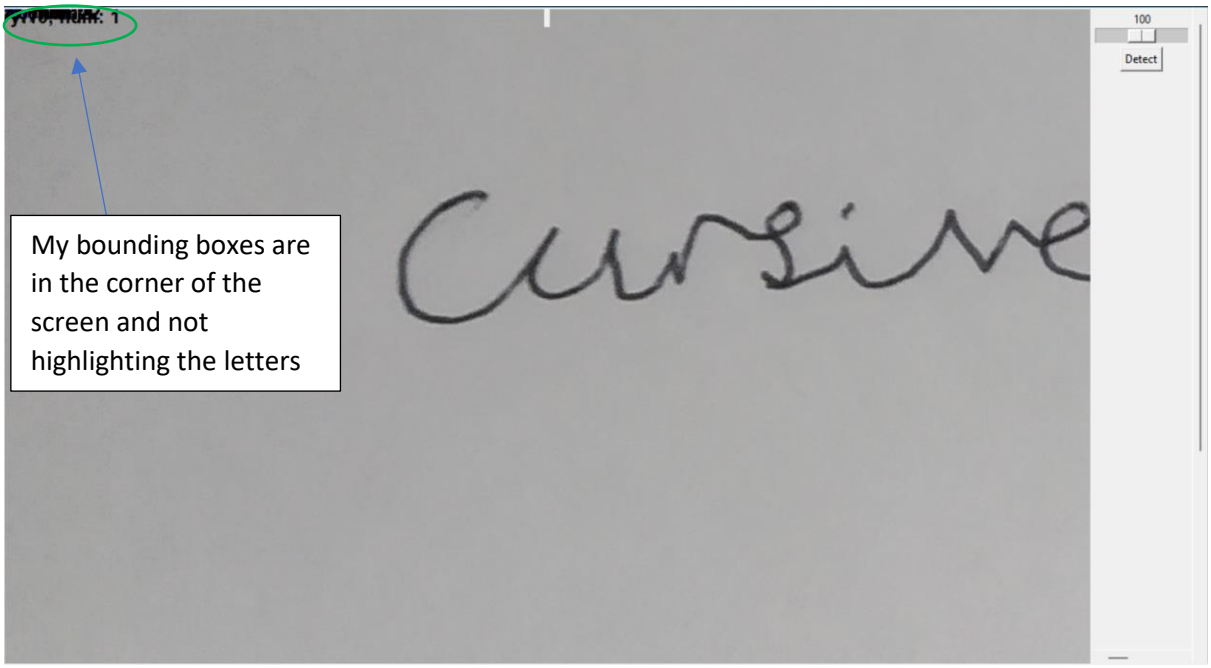
I imported my program and invoked the **main()** method that finds and saves contours into images and also outputs

scale_factor is a global variable that stores the value of **slider1** (zoom slider). I multiply the

My project: Handwriting recognition research project

My output when I run the new code for Tkinter:

When I run my code, it generates an erroneous output for me/



I think I know the problem with the current state of this code. Right now the code is very static because the coordinates of the bounding boxes are pre-set and do not change with the dimensions of the image (i.e., when using the zoom slider). I need to make the bounding boxes plotting more dynamic, meaning it is constantly adjusted to facilitate the dimensions of the image.

I have made an additional function that I will call inside zoom() which is similar to detect() (but it doesn't need import detect_letter)

```
def resize_bounding_with_zoom():
    global coor_info # I have used coor_info from detect as a global variable so I can use it throughout the program
    global scale_factor
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0][0]*scale_factor/100)) # x coordinate of bounding box after scale factor
        y_coor = int(round(coor_info[x][0][0][1]*scale_factor/100)) # y coordinate of bounding box after scale factor
        x_coor_w = int(round(coor_info[x][0][1][0]*scale_factor/100)) # x + w coordinate of bounding box after scale factor
        y_coor_h = int(round(coor_info[x][0][1][1]*scale_factor/100)) # # y + h coordinate of bounding box after scale factor
        print(x_coor, y_coor, x_coor_w, y_coor_h)
        canvas_image.create_rectangle(x_coor, y_coor, x_coor_w, y_coor_h, outline = "blue", tags = "rect")
        text_to_print = "x:{}, y:{}, num: {}".format(x_coor_w, y_coor_h, coor_info[x][1])
        canvas_image.create_text(x_coor_w, y_coor_h, text = text_to_print, fill = "black", font=('Helvetica 15 bold'), tags = "rect")
```

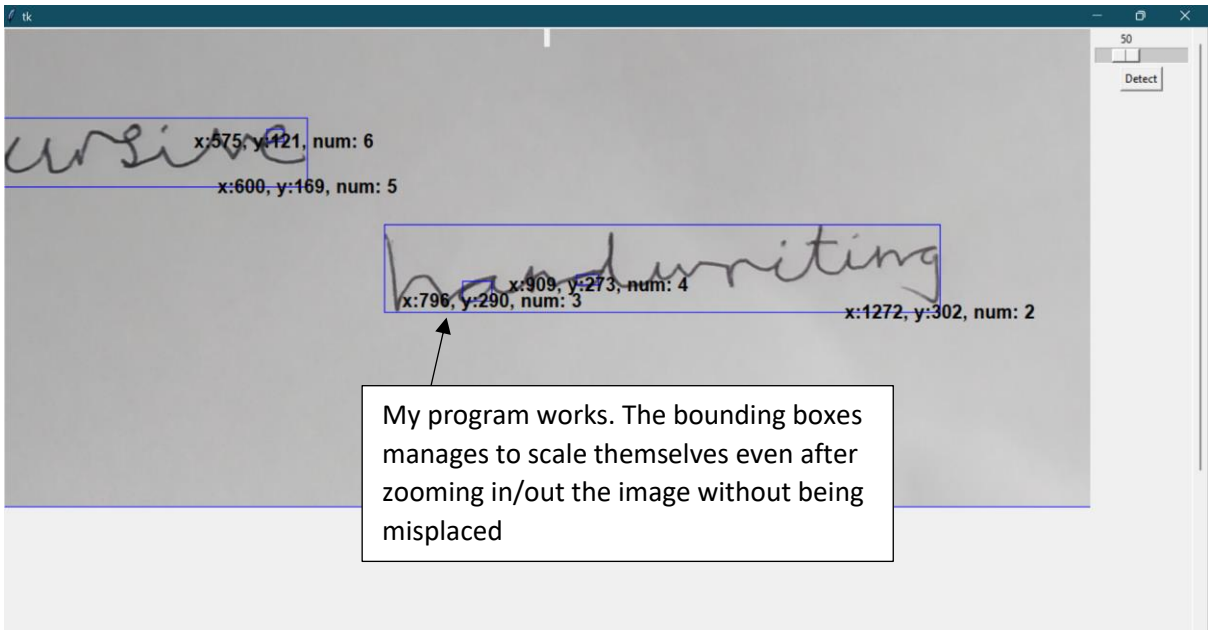
My project: Handwriting recognition research project

My invocation of resize_bounding_with_zoom() inside zoom():

```
def zoom(event):
    global newImage
    global scale_factor
    scale_factor = slider1.get() # this .get() method receives the input that the user gives for Scale()
    width = int(np.shape(imageOpen)[1]*scale_factor/100) # I change the width of the original image using scale factor
    height = int(np.shape(imageOpen)[0]*scale_factor/100) # I change the height of the original image using scale factor
    if width == 0:
        width = 1
    if height == 0:
        height = 1
    imageResize = imageOpen.resize((width, height)) # resizing image based on new width and height
    newImage = ImageTk.PhotoImage(image = imageResize)
    canvas_image.itemconfig(image_plot, image = newImage) # .itemconfig is used to update imageplot which deals with displaying images (in
    canvas_image.delete("rect")
    resize_bounding_with_zoom()
```

After the user uses the zoom slider, the coordinates of the bounding boxes are updated (based on scale factor) and all previous bounding boxes are removed.

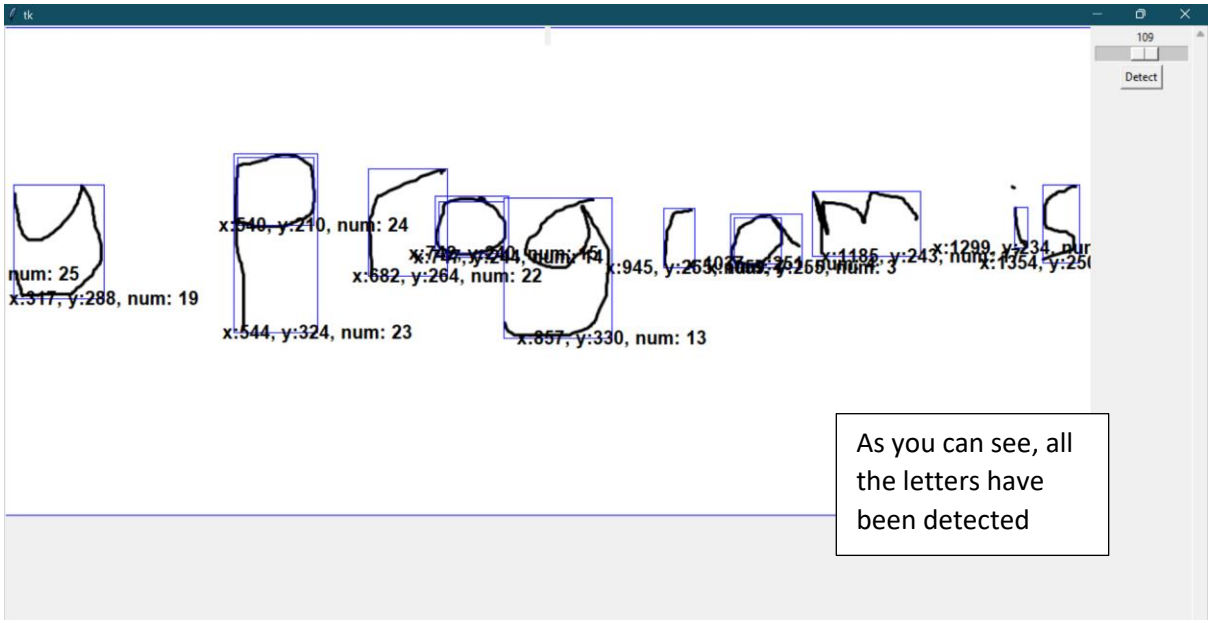
My output for Tkinter with this addition to code:



My program works. The bounding boxes manages to scale themselves even after zooming in/out the image without being misplaced

The above input image of cursive handwriting is a bad example of letter detection (it will not work in my program anyways), so I will demonstrate it with a better input image.

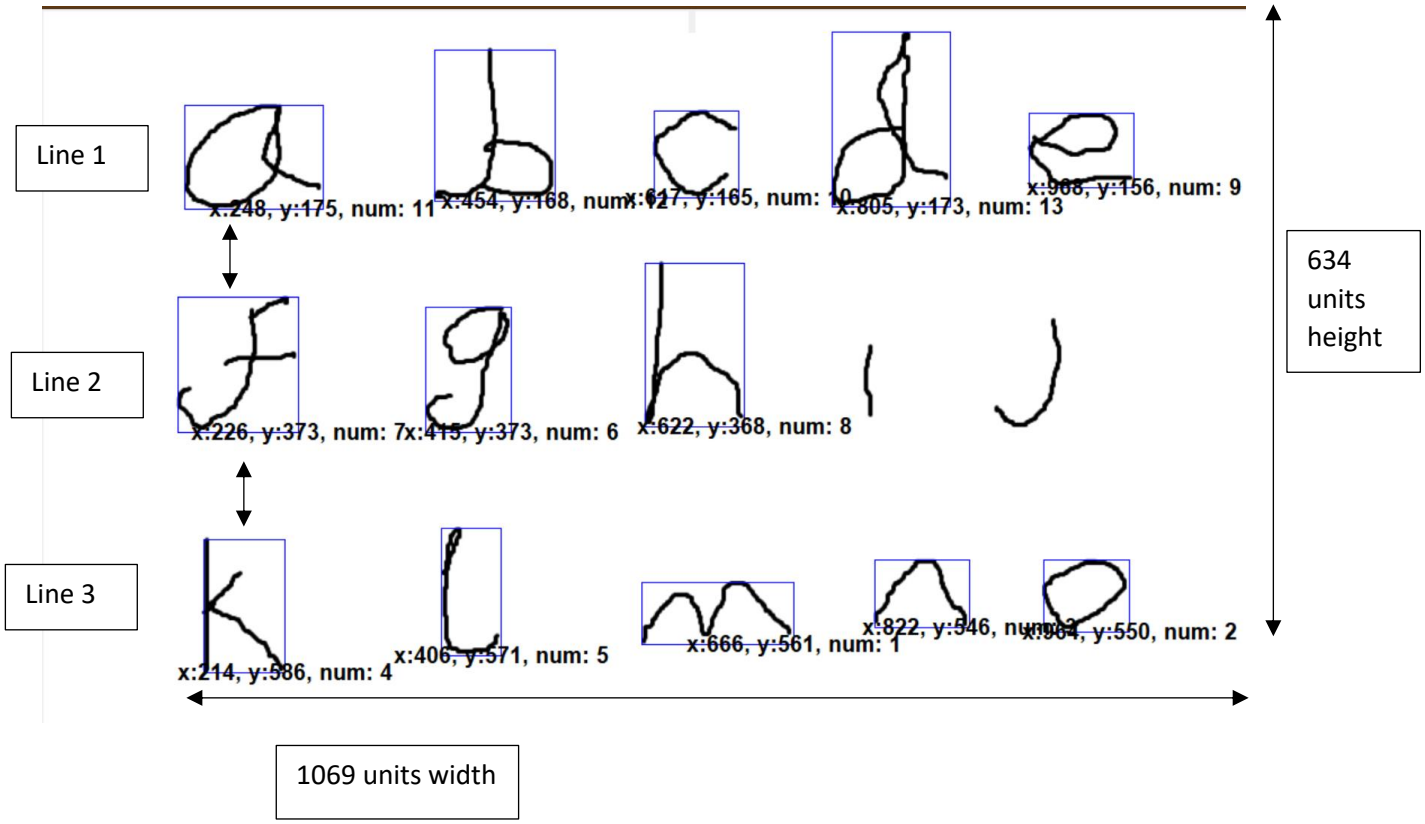
My project: Handwriting recognition research project



Section 3.3.6) Making partitioning algorithm for splitting letters into rows

I will now code my **points 4 and 5 from my Design section 2.2.2** using Tkinter coordinates for reference.

This is a sample of my image when I run my Tkinter program (**testGUI.py**):



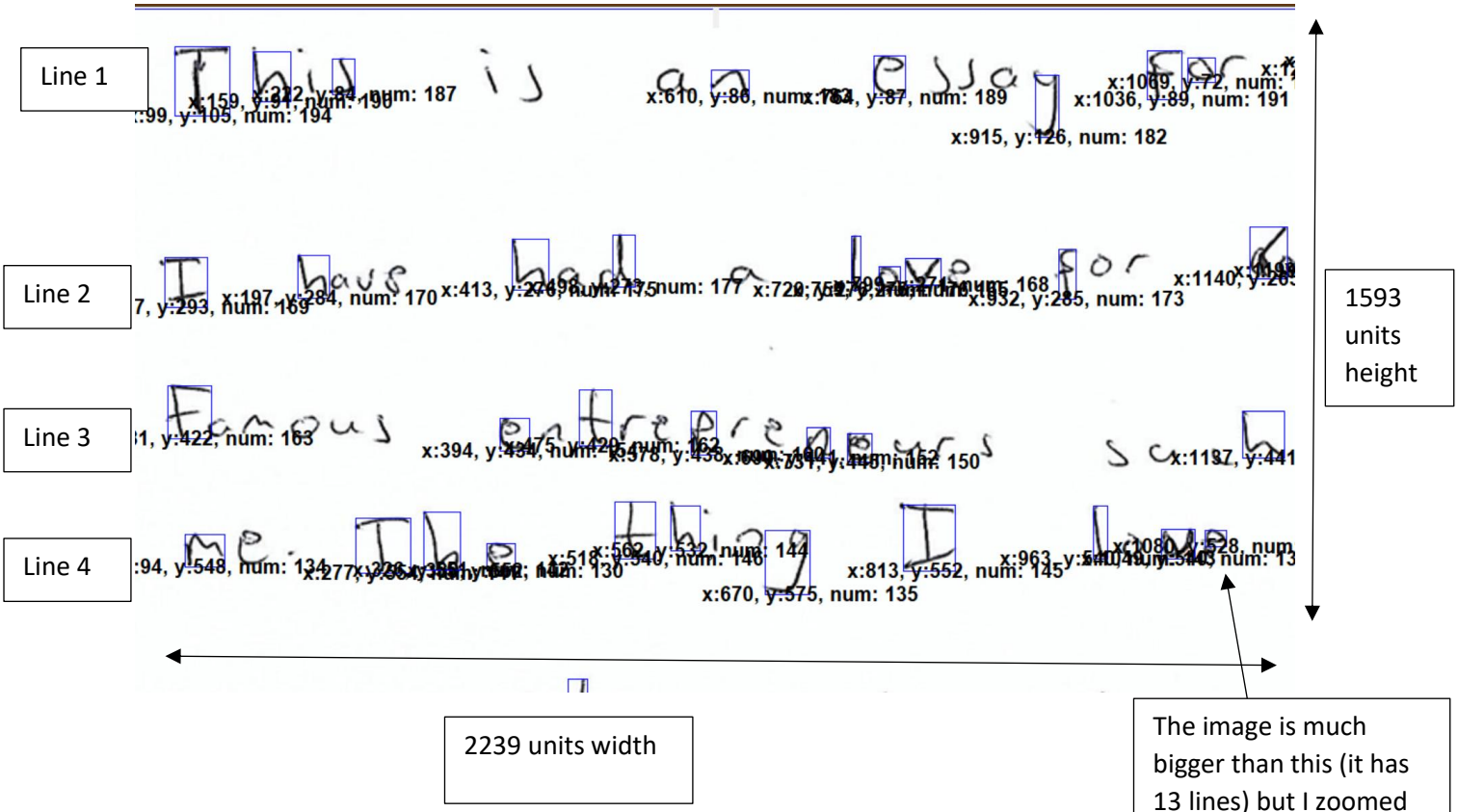
My project: Handwriting recognition research project

I will use my Tkinter program and sample images (with multiple lines of handwritten words) to determine the average gap between each line in any given image file. I will use this to partition my lines of letters, into arrays of coordinates.

In the image above, difference between the average y-coordinate of line 1, which is 167.4 (sum of all y-coordinates of line 1 letters/5) and average y-coordinate of line 2, which is 371.33(sum of all y-coordinates of line 1 letters/3) is 203.93. Now, this information alone is meaningless to me because this is the gap size for an image of 1069x634 specifically so we cannot generalise it for all image dimensions. To generalise this (so we can use to partition all images' rows) we need to divide 203.93 by the height of the image (634) which is 0.32, meaning that for any image of height x, if the gap between two y- coordinate coordinates is bigger than 0.32*x then both letters (for which the y-coordinate difference is worked out) are part of different lines. However I feel that this particular sample is a niche scenario because this image is big and only has 3 lines (evenly spaced) so the gap/height(of image) is going to output a higher percentage.

For my stakeholders, they write a lot of lines of assignments and homework, so the gap/height fraction for lines of their input will be lower.

To test a real life scenario out, I used this sample image:



For line 1 to line 2: $\frac{gap}{total\ height} = \frac{293-176}{1593} = 0.073$

For line 2 to line 3: $\frac{gap}{total\ height} = \frac{433.33-293}{1593} = 0.088$

For line 3 to line 4: $\frac{gap}{total\ height} = \frac{554-433.33}{1593} = 0.0757$

My project: Handwriting recognition research project

Using the above information, the minimum threshold I will have for my gap between any given adjacent y-coordinate values is 0.07.

I will write the code for my lines' partitioning:

```
def bubbleSort_sort_row(list1): # I will perform bubble sort on my array
    while True: # while loop doesn't stop until there is no adjacency swap
        swap = False
        for x in range(0, len(list1)-1):
            if list1[x][0][1][1] > list1[x+1][0][1][1]: # if the adjacent (to the right) element is smaller than the current then swap
                swap = True
                temp = list1[x]
                list1[x] = list1[x+1]
                list1[x+1] = temp # elements swapped
        if swap == False:
            return list1 # if there is no swaps in the list anymore, then bubble sort is complete, return new list
def main(coor_info, height): #I have made a function that takes the list of coordinates from detect_letter.py as a parameter
    y_sorted = bubbleSort_sort_row(coor_info) # first we need to bubble sort the coordinates so adjacent elements can be compared
    # note the relative position inside a line doesn't matter yet (we will rearrange that once we work out our arrays of each line)
    rows_sorted = []
    temp = 0 # I will use temp in slicing of my array to mark the first element of a new line
    for x in range(0, len(y_sorted)-1):
        if abs(y_sorted[x][0][1][1] - y_sorted[x+1][0][1][1])/height > 0.07: # if the difference between two y-coordinates (adjacent)/height > 0.07 means there is a new line
            rows_sorted.append(y_sorted[temp: x + 1]) # a new row is added with slicing between first index of the previous line (temp) and current index
            temp = x+1 # the current index becomes the new temp
        if x == len(y_sorted)-2: # I noticed during testing that the last line of the image was not saved, so this is an if statement to fix that
            rows_sorted.append(y_sorted[temp: x+2])
    return rows_sorted # return sorted list
```

This is bubble sort but just with y coordinates so I can later compare adjacent y-coordinates from smallest to biggest (to work out where a line partitions)

I followed my algorithm that I wrote in **section 2.2.2**, to partition lines and output a new array based on this (row_sorted)

Before starting testing, I will try a sample image to see if my program works as intended.

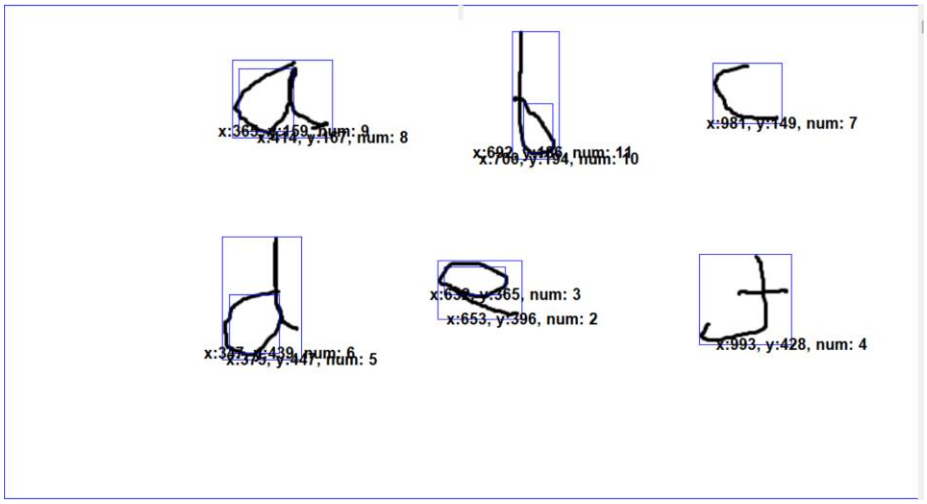
To test my python file **row_sorting.py**, I put imported **row_sorting** in my Tkinter program and used **coor_info** (output of **detect_letters.main()**) in my **detect()** function

```
def detect():
    global coor_info # I have used coor_info as a global variable so I can use it throughout the program
    import detect_letters # importing my program detect_letters
    coor_info = detect_letters.main("just_two_lines.png") # coordinate information of letters
    height = np.shape(Image.open("just_two_lines.png"))[0]
    import row_sorting
    output = row_sorting.main(coor_info, height)
    print("output: " + str(output))
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0][0]*scale_factor/100)) # x coordinate of bounding box
        y_coor = int(round(coor_info[x][0][0][1]*scale_factor/100)) # y coordinate of bounding box
        x_coor_w = int(round(coor_info[x][0][1][0]*scale_factor/100)) # x + w coordinate of bounding box
        y_coor_h = int(round(coor_info[x][0][1][1]*scale_factor/100)) # # y + h coordinate of bounding box
```

My code to test **row_sorting.py**

This is my sample image:

My project: Handwriting recognition research project



I want my **output** to have a length of 2 (because there are 2 rows of handwritten data) have coordinate information for num = 9, 8, 11, 10, 7 in the first row and in the second row I want coordinate information for num = 6, 5, 3, 2, 4.

My output:

The length of output is 3 meaning it recorded 3 rows, which I understand because the program counts the whole image as a bounding box as well so hence why there is a 3rd row

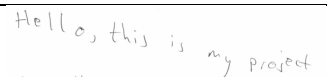
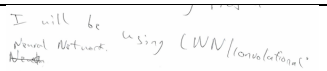
```
In [4]: len(output)
Out[4]: 3

In [5]: output[0]
Out[5]:
[[[(584.0, 48.0), (641.3333333333334, 97.33333333333333)], 7],
 [[(193.33333333333334, 52.0), (238.66666666666666, 104.0)], 9],
 [[(188.0, 45.333333333333336), (270.6666666666667, 109.33333333333333)], 8],
 [[(428.0, 81.33333333333333), (452.0, 121.33333333333333)], 11],
 [[(418.6666666666667, 21.333333333333332),
   (457.3333333333333, 126.66666666666667)],
  10]]

In [6]: output[1]
Out[6]:
[[[(362.6666666666667, 216.0), (413.3333333333333, 238.66666666666666)], 3],
 [[(357.3333333333333, 210.66666666666666),
   (426.6666666666667, 258.6666666666667)],
  2],
 [[(573.3333333333334, 205.33333333333334), (649.3333333333334, 280.0)], 4],
 [[(185.33333333333334, 238.66666666666666),
   (226.66666666666666, 286.6666666666667)],
  6],
 [[(180.0, 190.66666666666666), (245.33333333333334, 292.0)], 5]]

In [7]: output[2]
Out[7]: [[[(0.0, 0.0), (824.0, 406.6666666666667)], 1]]
```

It categoried num = 7, 9, 8, 11, 10 as first row which is what I wanted

My program categoried num = 3, 2, 4, 6, and 5 as second row which is what I wanted

My program is working well and not producing erroneous results so far. I will now do testing on **row_sorting.py** using my test data from **section 2.7.2**:

Test	Input data/input code	Desired output	Actual output	Pass/Fail	Explanation/justification
I will test whether my program output the	Hello, this is my project I will be using CNN (Convolution Network)	I want an output of 3 rows where	I get 3 rows (excluding an additional row	Pass	I expected this test to pass because I did some preliminary testing with similar data (see above)

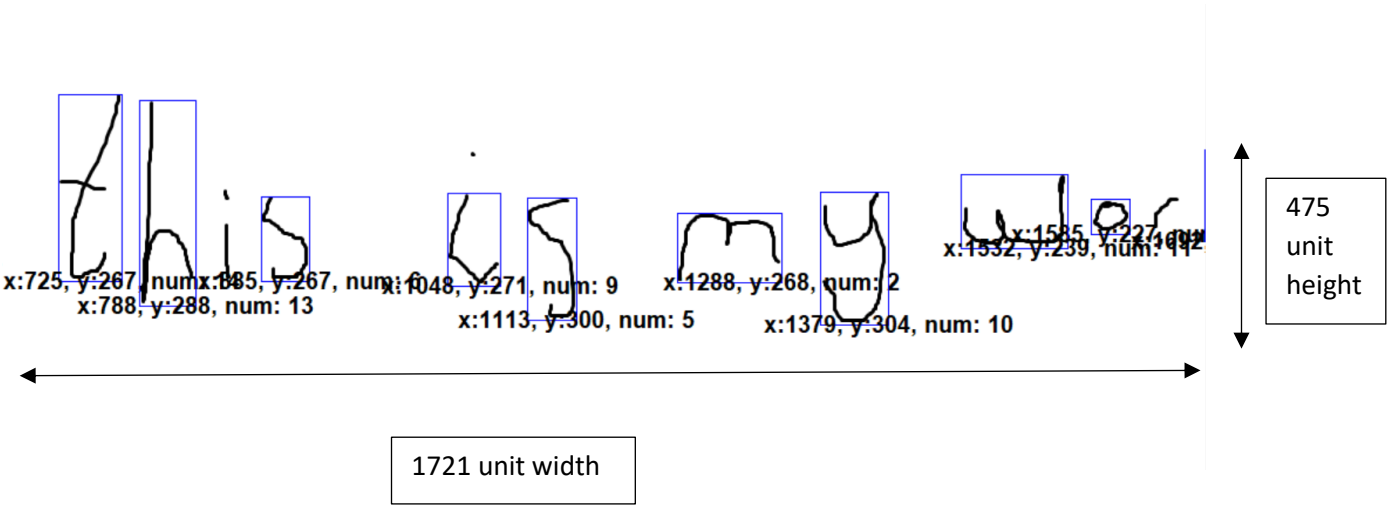
correct number of rows and each letter is correctly placed in its respective row		each row has its respective letters	but that is because one the bounding boxes is the whole image so it doesn't count)		and my program managed to sort that into rows correctly.
I will try a bit more difficult image (very slanted line of handwritten data) for sorting into rows, so I can observe how my program reacts to this image		I want an output of 1 row where the row has all its respective letters	I got 2 rows for this image	Fail	I anticipate a failed test for this image because the threshold for $\frac{gap}{total\ height}$ I used in my code was 0.07 which is means the gap between two y-coordinates for letters doesn't have to be big for it to register multiple lines. Since the line of writing in the input is very slanted, the difference of y-coordinates for letters exceeded 0.07 and hence multiple rows were registered.
I will try and put some erroneous data and observe how my program reacts to this.		Ideally, I would like 2 rows to be output	I got 4 rows as output for this image	Fail	I definitely expected this test to fail because this image is firstly hard to read and detect letters, and secondly, has very slanted lines so I did not expect any correct outputs.

Section 3.3.7) Making partitioning algorithm for splitting rows into words

My project: Handwriting recognition research project

Now that I finished sorting out my rows, I must now use x-coordinates of each letter within each row of any image and put them into their respective word formation so a sentence can be formed.

Using a similar math as I used for calculating a minimum threshold for row gaps, I will use sample images and my Tkinter program to calculate a minimum threshold for word gaps.



Unlike **row_sorting.py,** I do not need various images of handwritten data to simulate "real life scenarios" because word spaces are usually uniform and it is very easy to measure.

To work out the threshold for word gaps, I need to work out the difference between x-coordinate of the last letter for each word and the first letter of its adjacent word and then divide it by the width of the image.

$$\frac{gap}{width}\ for\ "this"\ and\ "is"\ = \frac{1048\ -\ 885}{1721}\ =\ 0.095$$

$$\frac{gap}{width}\ for\ "is"\ and\ "my"\ = \frac{1288\ -\ 1113}{1721}\ =\ 0.101$$

$$\frac{gap}{width}\ for\ "my"\ and\ "work"\ = \frac{1532\ -\ 1379}{1721}\ =\ 0.090$$

Using the three threshold values, the average threshold I will use is 0.095.

My project: Handwriting recognition research project

My code for grouping words in rows is:

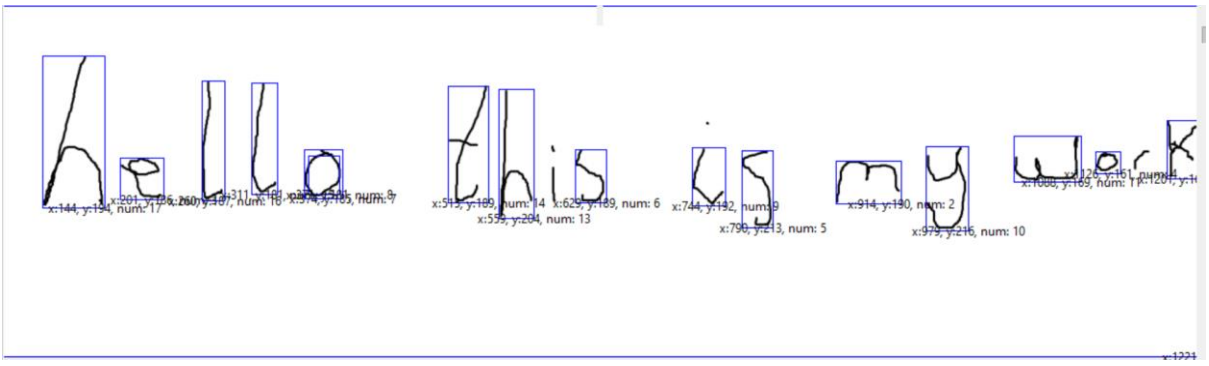
```
def bubbleSort_sort_column(list1): # I will perform bubble sort on my array
    while True: # while loop doesn't stop until there is no adjacency swap
        swap = False
        for x in range(0, len(list1)-1):
            if list1[x][0][1][0] > list1[x+1][0][1][0]: # if the adjacent (to the right) x coordinate is smaller than the c
                swap = True
                temp = list1[x]
                list1[x] = list1[x+1]
                list1[x+1] = temp # elements swapped
        if swap == False:
            return list1 # if there is no swaps in the list anymore, then bubble sort is complete, return new list
def main(sorted_rows, width): #I have made a function that takes the list of coordinates from row_sorting.py as a parameter
    new_sorted_individual_row = []
    for x in range(0, len(sorted_rows)):
        a = bubbleSort_sort_column(sorted_rows[x]) # within each row, the list is sorted by x coordinates
        new_sorted_individual_row.append(a) # new list is appended to new_sorted_individual_row
    #return new_sorted_individual_row
    words_list_line_by_line = [] # this will store the rows and the word in the image
    for x in range(0, len(new_sorted_individual_row)):
        temp = 0  # I will use temp in slicing of my array to mark the first element of a new word
        word_row_list = [] # this stores the words made within each row
        for y in range(0, len(new_sorted_individual_row[x])-1):
            print(new_sorted_individual_row[x][y][1])
            if abs(new_sorted_individual_row[x][y][0][1][0] - new_sorted_individual_row[x][y+1][0][1][0])/width > 0.095: # if the difference between two adjacent x-coordinates/width > 0.095, form a word
                word_row_list.append(new_sorted_individual_row[x][temp:x]) # add the elements between temp and current index to word_row_list (new word)
                temp = x+1 # update temp to index of first letter in a new word
        words_list_line_by_line.append(word_row_list) # add each row of words to words_list_line_by_line
    return words_list_line_by_line # output
```

This is bubble sort but just with x coordinates so I can later compare adjacent x-coordinates from smallest to biggest (to work out where a word partitions)

First of all, I will pass each index of my **sorted_rows** into **bubbleSort_column()** so each row will have sorted elements (in terms of x coordinates), I will save the outputs of the bubbleSort in a new array **new_sorted_individual_row**

Here I iterate a loop through **new_sorted_individual_row** and find the difference between each adjacent pairs of nodes in each row. If so, form a word and save it is **word_row_list,** at the end of the iteration of x, save **word_row_list** into **words_list_line_by_line**.

To make sure my code is working properly, I will test it on one sample image:



The output I should get for **words_list_line_by_line** is:

It should have two rows, first one having a length of 5 (for 5 words), and second one having 0.

The output I get:

My project: Handwriting recognition research project

```
In [7]: words_list_line_by_line
Out[7]: [[[], []], []]
```

This is not the output I wanted, this output for words_list_line_by_line is empty, this means that nothing is passing in my if statement for **new_sorted_individual_row**

To diagnose the problem, I have put print statement within the nested for loops to check for the problem.

```
for x in range(0, len(new_sorted_individual_row)):
    temp = 0  # I will use temp in slicing of my array to mark the first element of a new word
    word_row_list = [] # this stores the words made within each row
    for y in range(0, len(new_sorted_individual_row[x])-1):
        print("num inside Loop: {}".format(new_sorted_individual_row[x][y][1]))
        if abs(new_sorted_individual_row[x][y][0][1][0] - new_sorted_individual_row[x][y+1][0][1][0])/width > 0.095: # if the difference betw
            print("entered inside if statement at x = {}, y = {}, num = {}".format(x,y,new_sorted_individual_row[x][y][1]))
            word_row_list.append(new_sorted_individual_row[x][temp:x]) # add the elements between temp and current index to word_row_list (ne
            temp = x+1 # update temp to index of first letter in a new word
    words_list_line_by_line.append(word_row_list) # add each row of words to words_list_line_by_line
```

Check if letters (using their identifier **num**) have entered the for loop

Check if letters (using their identifier **num**) have entered the if statement.

```
In [28]: a = main(output, 1721)
num inside loop: 17
num inside loop: 3
num inside loop: 16
num inside loop: 15
num inside loop: 8
num inside loop: 7
entered inside if statement at x = 0, y = 5, num = 7
num inside loop: 14
num inside loop: 13
num inside loop: 6
entered inside if statement at x = 0, y = 8, num = 6
num inside loop: 9
num inside loop: 5
entered inside if statement at x = 0, y = 10, num = 5
num inside loop: 2
num inside loop: 10
entered inside if statement at x = 0, y = 12, num = 10
num inside loop: 11
num inside loop: 4
```

So, I can deduce that my if statement parameter was correct. The program enters the if statement when the current letter is the end of its word

As I was checking over my code, I found out the huge mistake I was making:

```
for x in range(0, len(new_sorted_individual_row)):
    temp = 0  # I will use temp in slicing of my array to mark the first element of a new word
    word_row_list = [] # this stores the words made within each row
    for y in range(0, len(new_sorted_individual_row[x])-1):
        print("num inside Loop: {}".format(new_sorted_individual_row[x][y][1]))
        if abs(new_sorted_individual_row[x][y][0][1][0] - new_sorted_individual_row[x][y+1][0][1][0])/width > 0.095: # if the difference betw
            print("entered inside if statement at x = {}, y = {}, num = {}".format(x,y,new_sorted_individual_row[x][y][1]))
            word_row_list.append(new_sorted_individual_row[x][temp:x]) # add the elements between temp and current index to word_row_list (ne
            temp = x+1 # update temp to index of first letter in a new word
    words_list_line_by_line.append(word_row_list) # add each row of words to words_list_line_by_line
```

I was using **x** for slicing of my array instead of **y**. Usually you would get an **Index error** if you input the wrong values as indexes however, since x was within array index bounds I failed to notice this

My project: Handwriting recognition research project

I changed the code to fix my error:

```
for x in range(0, len(new_sorted_individual_row)):
    temp = 0  # I will use temp in slicing of my array to mark the first element of a new word
    word_row_list = [] # this stores the words made within each row
    for y in range(0, len(new_sorted_individual_row[x])-1):
        print("num inside loop: {}".format(new_sorted_individual_row[x][y][1]))
        if abs(new_sorted_individual_row[x][y][0][1][0] - new_sorted_individual_row[x][y+1][0][1][0])/width
            print("entered inside if statement at x = {}, y = {}, num = {}".format(x,y,new_sorted_individua
            print("slicing {}".format(new_sorted_individual_row[x][temp:y+1]))
            word_row_list.append(new_sorted_individual_row[x][temp:y+1]) # add the elements between temp a
            temp = y+1 # update temp to index of first letter in a new word
    words_list_line_by_line.append(word_row_list) # add each row of words to words_list_line_by_line
return words_list_line_by_line # output
```

I changed the temp and index slicing to **y** instead of **x**

And the output for my program is:

Index	Type	Size	Value
0	list	6	[[[...], 17], [[...], 3], [[...], 16], [[...], 15], [[...], 8], [[...] ...
1	list	3	[[[...], 14], [[...], 13], [[...], 6]]
2	list	2	[[[...], 9], [[...], 5]]
3	list	2	[[[...], 2], [[...], 10]]

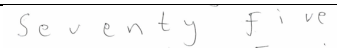
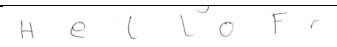
I used the Spyder variable explorer and I managed to get the output that I wanted. I have 4 arrays (one representing a word) and each array contains the coordinate of its corresponding letter

My final code for **laying_out_words.py**:

My project: Handwriting recognition research project

```
def bubbleSort_sort_column(list1): # I will perform bubble sort on my array
    while True: # while loop doesn't stop until there is no adjacency swap
        swap = False
        for x in range(0, len(list1)-1):
            if list1[x][0][1][0] > list1[x+1][0][1][0]: # if the adjacent (to the right) x coordinate is smaller than the current then swap
                swap = True
                temp = list1[x]
                list1[x] = list1[x+1]
                list1[x+1] = temp # elements swapped
        if swap == False:
            return list1 # if there is no swaps in the list anymore, then bubble sort is complete, return new list
def main(sorted_rows, width): #I have made a function that takes the list of coordinates from row_sorting.py as a parameter
    new_sorted_individual_row = []
    for x in range(0, len(sorted_rows)):
        a = bubbleSort_sort_column(sorted_rows[x]) # within each row, the list is sorted by x coordinates
        new_sorted_individual_row.append(a) # new list is appended to new_sorted_individual_row
    #return new_sorted_individual_row
    words_list_line_by_line = [] # this will store the rows and the word in the image
    for x in range(0, len(new_sorted_individual_row)):
        temp = 0  # I will use temp in slicing of my array to mark the first element of a new word
        word_row_list = [] # this stores the words made within each row
        for y in range(0, len(new_sorted_individual_row[x])-1):
            print("num inside loop: {}".format(new_sorted_individual_row[x][y][1]))
            if abs(new_sorted_individual_row[x][y][0][1][0] - new_sorted_individual_row[x][y+1][0][1][0])/width > 0.08: # if the difference between two adjacent x-coordinates/width > 0.095, form a word
                print("entered inside if statement at x = {}, y = {}, num = {}".format(x,y,new_sorted_individual_row[x][y][1]))
                #print("slicing {}".format(new_sorted_individual_row[x][temp:y+1]))
                word_row_list.append(new_sorted_individual_row[x][temp:y+1]) # add the elements between temp and current index to word_row_list (new word)
                temp = y+1 # update temp to index of first letter in a new word
            if y == len(new_sorted_individual_row[x])-2: # I noticed during testing that the last line of the image was not saved, so this is an if statement to fix that
                word_row_list.append(new_sorted_individual_row[x][temp: y+2])
        words_list_line_by_line.append(word_row_list) # add each row of words to words_list_line_by_line
    return words_list_line_by_line # output
```

My program is working well and not producing erroneous results so far. I will now do testing on **laying_out_words.py** using my test data from **section 2.7.2**:

Test	Input data/input code	Desired output	Actual output	Pass/Fail	Explanation/justification
I will test a standard image (like the one used above) to verify my program is outputting word arrays correctly		The final list will have 1 row with its length being 5 (because there are 5 words)	The final list does have 1 row with its length being 5	Pass	I expected this test to pass because I used a similar image in my preliminary testing above.
I will test a difficult input this time to see what my program outputs (and whether its erroneous or not)		The final list will have 1 row with its length being 2 (because there are 2 words)	The final list has 1 row with its length being 3	Fail	The reason for this failed test is the fact that the gap between “v” and “e” is too big in “seventy” so it recognised it as a word partition.
I will enter		The final list	The final	Fail	I definitely expected my program to fail

erroneous data to observe what the program will output as a result		will have 1 row with its length being 2 (because there are 2 words)	list has 1 row with its length being 7		on this test because this input image is too difficult to correctly decompose, and the gaps between each word are just too big.

Section 3.3.8) Review

Work I have done

In this section I mainly coded 5 algorithms:

1. Detecting letters in an input image and saving it as standalone image files (for later image processing and analysis). The python file is called **detect_letters.py**
2. (Not included as part of my key features of solution (**section 1.5**)) cropping and placing letter images (outputs of **detect_letters.py**) at the centre of a larger blank white images so that the letter doesn't look too zoomed out, and potentially lose quality. The python file is called **spacing_for_letter.py**.
3. Making the thickness of the letters bigger so the images don't appear faint so it is crisp and clear for the neural network to recognise. The python file is called **thickness_of_letter_enlarged.py.**
4. Partitioning letters into rows using y-coordinates of letters so that the structure of the handwritten image can be modelled in text format. The python file is called **row_sorting.py.**
5. Partitioning letters (that are in rows) into words by using their x-coordinates and a fixed threshold value (which I worked would be 0.09 for me). The python file is called **laying_out_words.py.**

I additionally made a Tkinter GUI for my testing purposes (for example, displaying **num** values for letters (unique identifiers that I assigned to letter images) on Tkinter window alongside bounding boxes, to monitor and observe the results of my partitioning algorithms (points 44 and 5))

Testing of my work

I have done all the necessary testing in Stage 2 (using test data that I made in **Section 2.7.2**) and I am satisfied with my test results for image processing.

Has it met my key features of my solution (**Section 1.5**)/success criteria (**Section 1.6**)?

Key features (neural network)	Met/Not Met	Explanation/Justification
Detect individual letters	Met	I used OpenCV contour detection to detect letters in an image. While it is not perfect (sometimes it doesn't detect some letters) it mostly works and is satisfactory enough to use in my final GUI
Make image contours thicker	Met	I made an algorithm using OpenCV contour detection (again) that adds additional layers of contour onto the letter and make it look thicker. As I found out, this does have varying degree of success on neural network success rate
Partition detected letters into rows	Met	I made this algorithm using outputs (coordinate arrays) of **detect_letters.py** and it works successfully in partitioning all letters into the structure of the handwritten image
Group detected letters to form words	Met	Once I partitioned the letters, I used the function's output array as input for this algorithm. I managed to completely copy the structure of my input image using both of these algorithms.

Steps to improve it

There is a lot of space for improvement in my image processing section. There is a lot of data validation that I looked over for my detection algorithm (for example making faint images sharper, so fainter letters can be detected too) which could have been beneficial for my project. I could have also looked deeper into thresholding algorithms and experimented more with its threshold parameters, so I could get the perfect image for letter detection.

Section 3.4) Stage 3: Graphical User Interface

Section 3.4.1) Reusing GUI made in Section 3.3.5 and allowing user input (with validation)

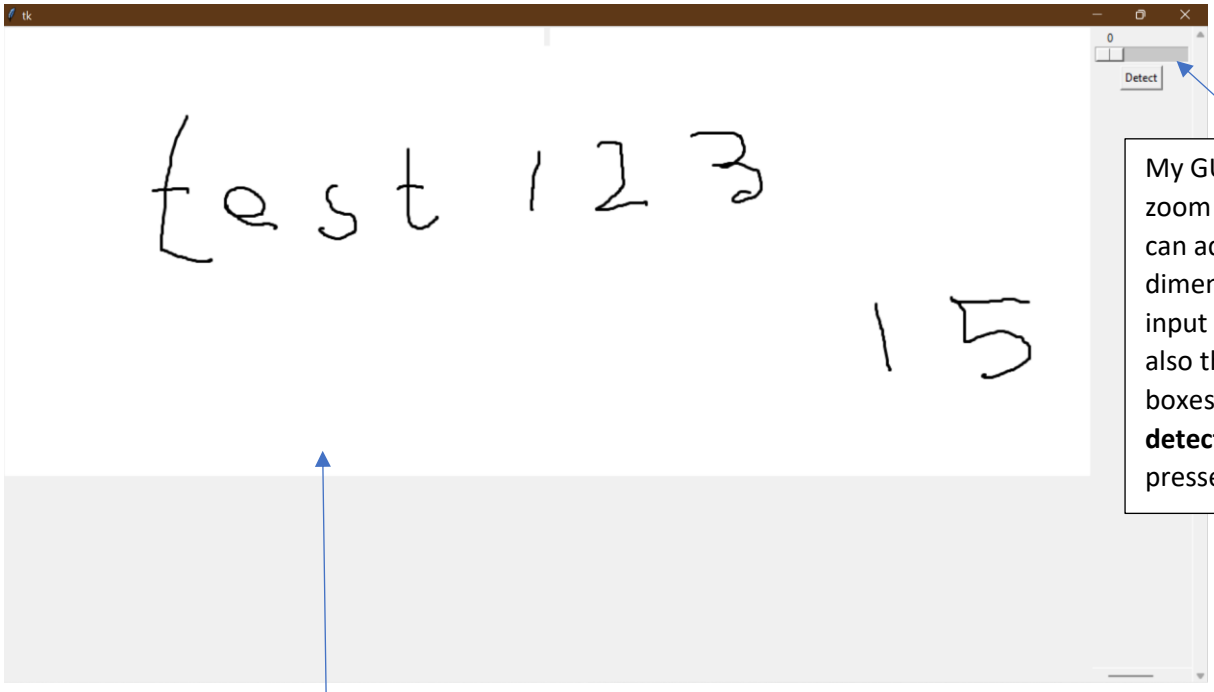
My project: Handwriting recognition research project

As it stands, I have all the necessary parts for letter recognition using user inputs. These parts are:

1) Neural Network – I have used Tensorflow which is very reliable and outputs accurate results based on inputs.
2) Image Processing methods for processing input:
    a. Detecting letters in an input image and saving it as standalone image files which can be used for letter recognition (after doing additional image processing)
    b. Making the thickness of the letters bigger so the images appear sharper and clearer for image recognision
    c. Mapping the letter images out using arrays to make them structurally identical to the input handwritten image (for example, partitioning out letters into lines/rows (if the handwritten image has lines/rows)) so that when the image data is converted to text format, it is a structural replica of the input handwritten image.

In Section 2.3, I said that I wanted my program to have an output window for showing the user input (handwritten image) and a zoom in/out functionality that the user can use to resize the output window. However, since I have already made that in my Tkinter program that I created from scratch for testing (testGUI.py) and explained about the process in Section 3.3.5, I will reuse those components in my final Tkinter GUI program.

Therefore, this is my GUI so far:



My GUI has a zoom slider which can adjust the dimensions of the input image and also the bounding boxes (if the **detect** button is pressed)

My GUI has an output window where the user can see their input image

First of all I need to add a way for the user to input their image. In my design of the GUI (section 2.3) I specifically said to make a dropdown menu bar that allows the user to open a file.

My project: Handwriting recognition research project

This is my implementation of a menu bar:

```
root = Tk() #defining the main window of the program
menu1 = Menu(root) # the Menu() defines a menu for me in the window
menu_dropdown = Menu(menu1, tearoff = 0)
menu_dropdown.add_command(label = "Open", command = open_file) # this is the actual button that will allow the user to input images
menu_dropdown.add_separator()
menu1.add_cascade(label = "File operations", menu = menu_dropdown) # this is the tab the user has to click on to get to the "Open" button
root.config(menu = menu1) # stating that the main window will have a menubar
```

open_file() is a function that I will call so the user can enter data

This is where I define a menu bar for **root**(window)

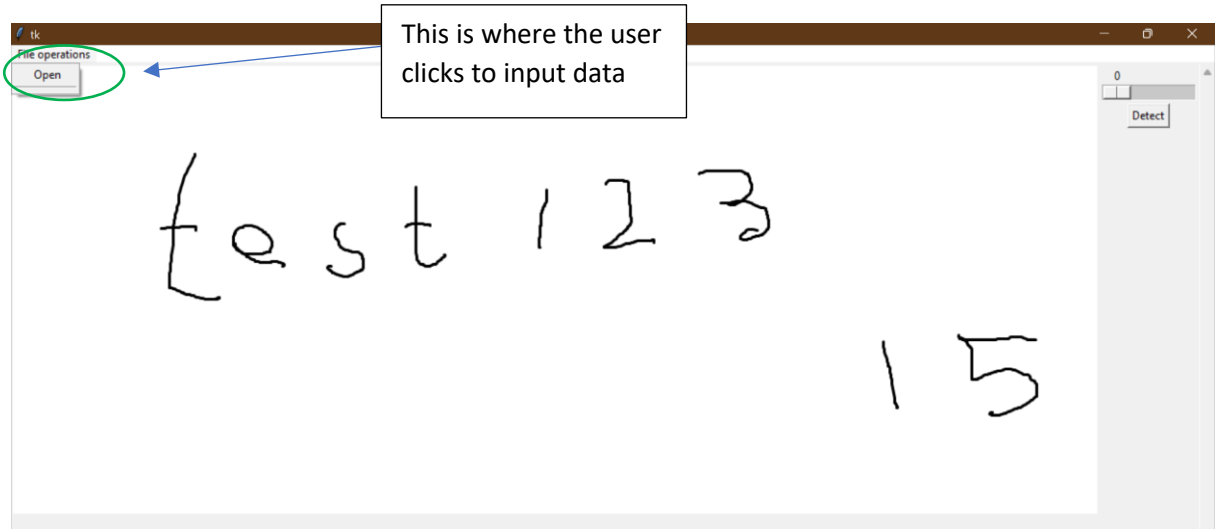
This is my open_file() function:

I have defined global variables so I can update image path parameters in the program for example for **detect_letters.py**, I will use **name_of_file**

```
def open_file():
    global name_of_file
    global image
    global imageOpen
    filetypes = (
        ("PNG file", '*.png'),
        ("JPEG/JPG file", '*.jpeg *.jpg') # these are the only types of files acceptable when using filedialog.askopenfile()
        )
    from tkinter import filedialog
    name_of_file = filedialog.askopenfilename( # this is a tkinter method that opens a window and allows the user to manually search their directories to open a file
        title = "Open a file",
        initialdir = "/",
        filetypes = filetypes # I have defined this above, PNG and JPEG/JPG are the only filetypes allowed
        )
    imageOpen = PIL.Open(name_of_file) # opening the file path that the user has input
    image = ImageTk.PhotoImage(image = imageOpen)
    canvas_image.itemconfig(imgtag, image = image) # displays the image onto canvas_image
    canvas_image.delete("rect")
```

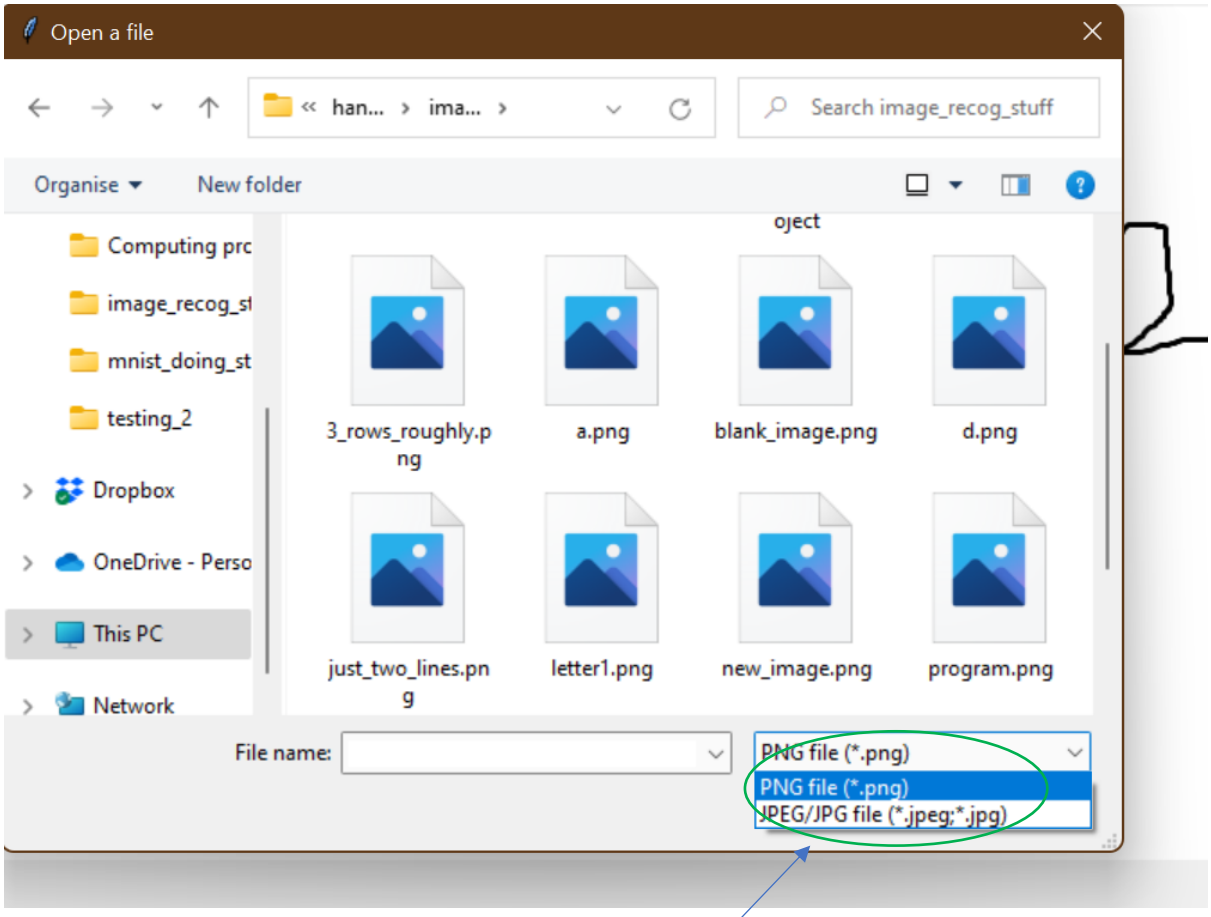
This is my user validation for this program. I have defined **filetypes** (which is a parameter of **filedialog.askopenfilename()**) where I can assign the acceptable file types. In my case, only png/jpeg/jpg are allowed

Result of the implementation:

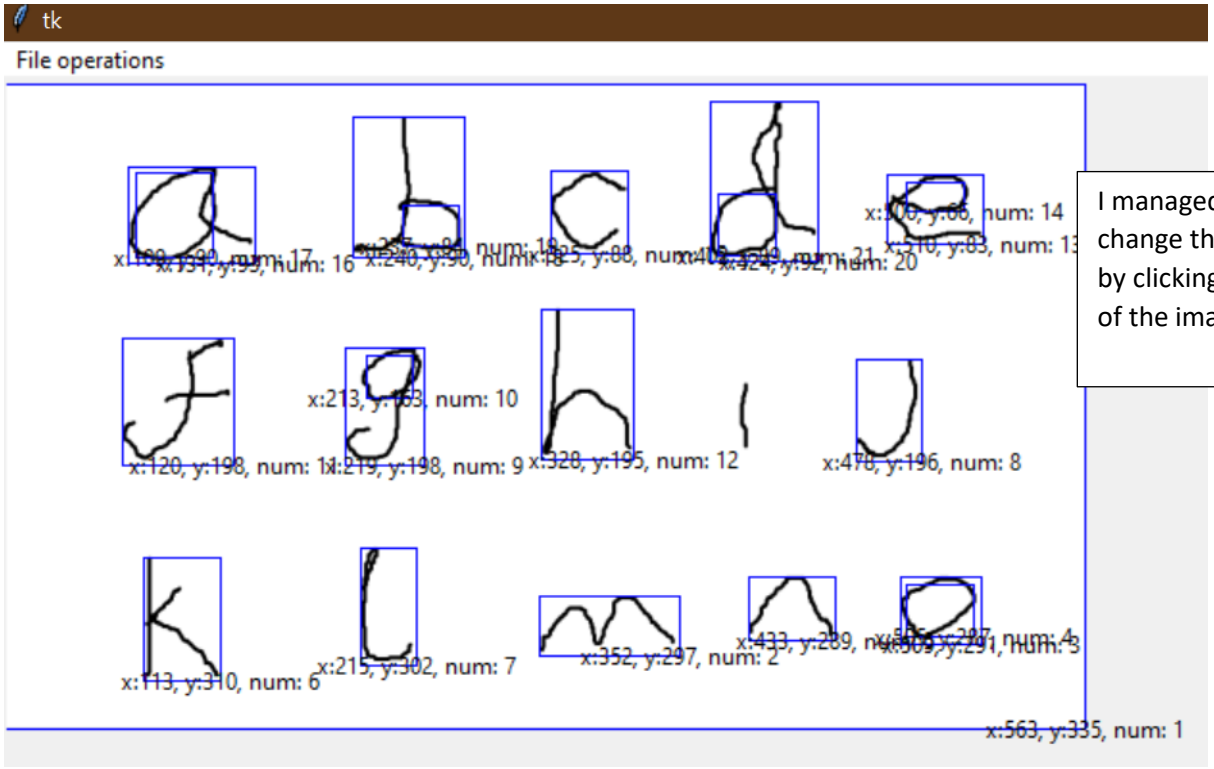


Once I click on “Open”, I get a menu to select my input image:

My project: Handwriting recognition research project



This is user validation. I have made sure that only .png, or .jpg/.jpeg can be selected



I managed to change the image by clicking on one of the images

Section 3.4.2) Making a delete button

The next thing I wanted to add to my neural network is a delete button to clear up the bounding boxes on the screen. This is so that the interface can look clean and not cluttered.

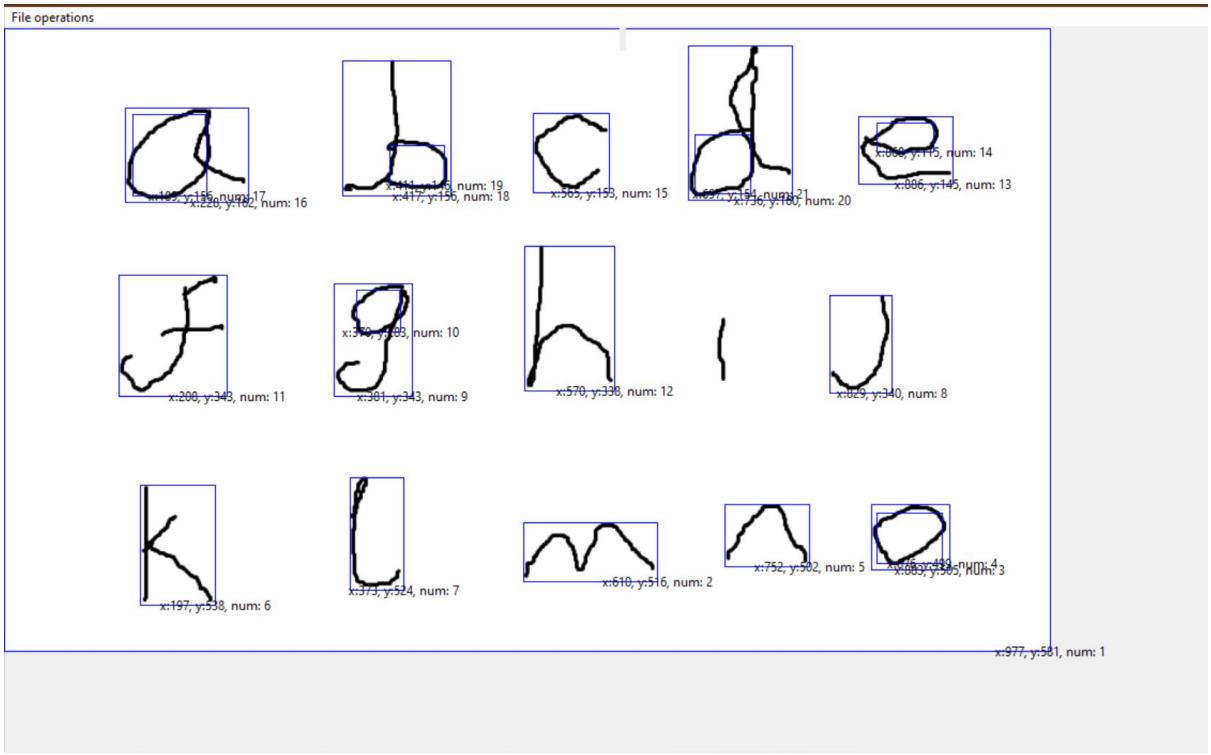
This is my implementation for the delete button:

```
def delete_canvas_rectangles():
    canvas_imagse.delete("rect")
delete_button = Button(root, text = "Delete", command = delete_canvas_rectangles)
```

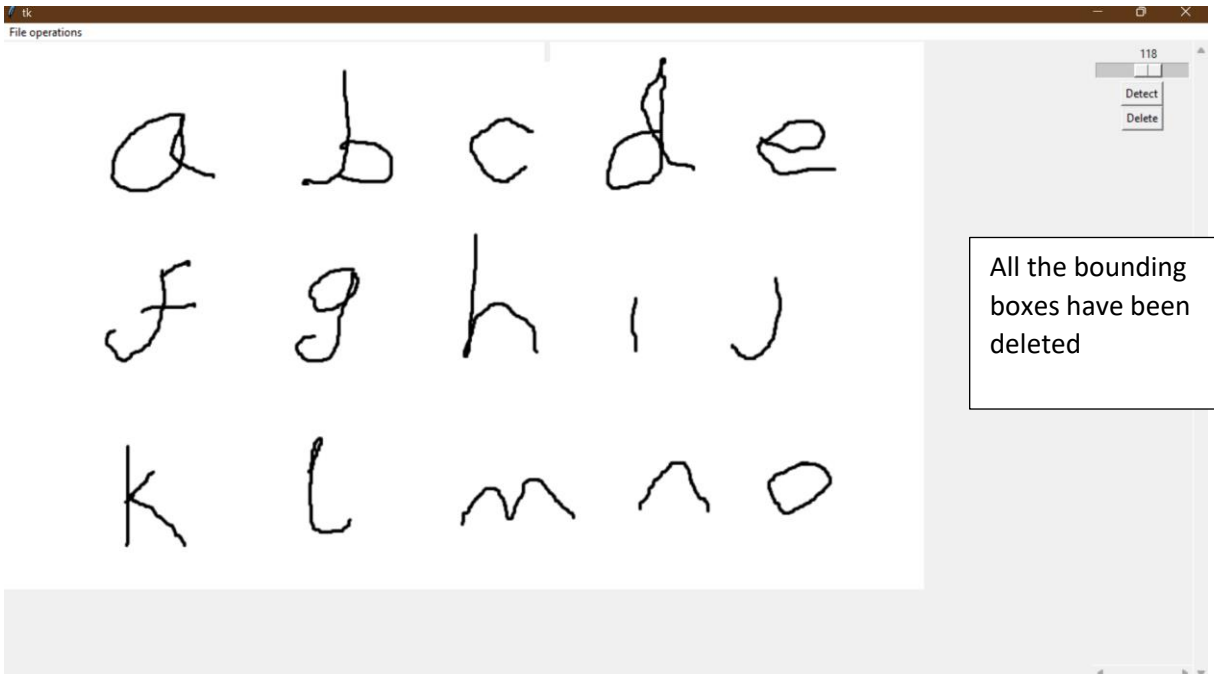
Every time I implement **canvas_image.create_text()** or **canvas_image.create_text()**, I set a parameter called **tags= "rect"**, where every text and rectangle widget will be given a tag. I can use the **canvas_image.delete("rect")** to delete every widget that has a tag of **"rect".**

Testing the delete button:

Before using delete button:



After using delete button:



All the bounding boxes have been deleted

Section 3.4.3) Making a recognition button (part 1)

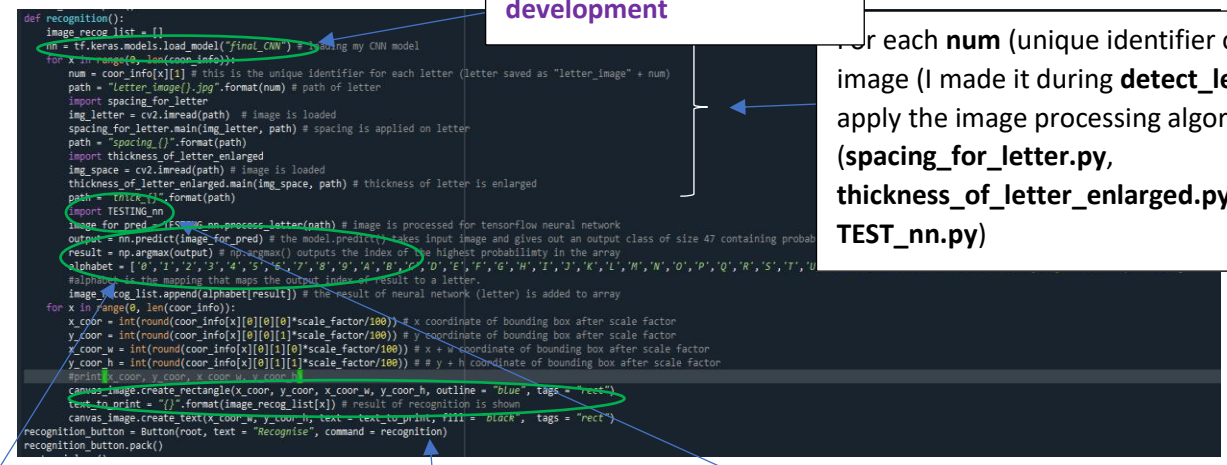
Now, I must design a function for image recognition using my Tensorflow model I made in Section 3.2.2.

My end goal for my image recognition button is to redirect to a second page where the output will be written in text format. However, first I need to verify that my neural network recognises letters in an image first, therefore my image recognition button will

My project: Handwriting recognition research project

temporarily call a function that out[illegible]ng letter image) as text for a bounding box. Here is my implen[illegible]

I load my neural network that I made in **stage 1 of development**



For each **num** (unique identifier of letter image (I made it during **detect_letter.py**)) apply the image processing algorithms (**spacing_for_letter.py**, **thickness_of_letter_enlarged.py**, **TEST_nn.py**)

After image processing I pass in the image into my neural network (**nn**). The result of the neural network is saved in **image_recog_list**

I haven't talked about **TESTING_nn.py** before, it is just the file of the image processing algorithm (using PIL) that I used in **stage 1 of development** to process image for Tensorflow. I will attach **TESTING_nn.py** code below

In a new for loop, I am displaying results of neural network recognition (that is stored in **image_recog_list**) for every bounding box

Before I run my code I also commented (temporarily) the **canvas_image.create_text()** mentions in **detect()** and **resize_bounding_with_zoom()**

My code for **TESTING_nn.py** (for reference above)

```
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image, ImageOps
#nn = tf.keras.models.load_model("final_CNN") # loading my CNN model
#from PIL import Image, ImageOps
def process_letter(img_path):
    image = Image.open(img_path) # opening an image using its path
    image_resized = image.resize((28,28)) # resizing image to 28 by 28 so that inverse and grayscale operations can be performed
    image_gray = image_resized.convert("L") # converts to grayscale
    image_inverse = ImageOps.invert(image_gray) # inverts the image so the background is black and foreground is white
    numpy_image = np.array(image_inverse) # converting to numpy array for reshaping it for CNN
    plt.imshow(numpy_image)
    plt.show()
    final1 = numpy_image.reshape(1, 28, 28, 1) # resizing array for CNN
    final1 = final1/255 # normalising the data so it is continuous
    return final1
```

When I clicked on my "Recognition" button I got the following error:

My project: Handwriting recognition research project

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\Sahil\AppData\Local\Programs\Python\Python39\lib\tkinter\__init__.py", line 1884, in __call__
    return self.func(*args)
  File "c:\users\sahil\desktop\computing project\handwriting-recognition-ann-master\image_recog_stuff\finalgui.py", line 124, in recognition
    thickness_of_letter_enlarged.main(img_space, path) # thickness of letter is enlarged
  File "C:\Users\Sahil\Desktop\computing project\handwriting-recognition-ann-master\image_recog_stuff\thickness_of_letter_enlarged.py", line 9, in main
    gray = cv2.cvtColor(x, cv2.COLOR_BGR2GRAY)
cv2.error: OpenCV(4.5.2) C:\Users\runneradmin\AppData\Local\Temp\pip-req-build-ttbyx0jz\opencv\modules\imgproc\src\color.cpp:182: error: (-215:Assertion failed) !_src.empty() in function
'cv::cvtColor'
```

The Python shell shows that the error occurs in the line **thickness_of_letter_enlarged.main(img_space, path)**, meaning that either my **path** is wrong or my **img_space** is invalid

To diagnose the problem, I put in print statements to manually verify if my paths for images are correct:

```
image_recog_list = []
for x in range(0, len(coor_info)):
    num = coor_info[x][1] # this is the unique identifier for each letter
    path = "letter_image{}.jpg".format(num) # path of letter
    print("path of original: {}".format(path))
    import spacing_for_letter
    img_letter = cv2.imread(path)  # image is loaded
    spacing_for_letter.main(img_letter, path) # spacing is applied on let
    path = "spacing_{}".format(path)
    print("path of spacing: {}".format(path))
    import thickness_of_letter_enlarged
    img_space = cv2.imread(path) # image is loaded
    thickness_of_letter_enlarged.main(img_space, path) # thickness of let
    path = "thick_{}".format(path)
    print("path of thick: {}".format(path))
    import TESTING_nn
    image_for_pred = TESTING_nn.process_letter(path) # image is processed
```

Print statements for diagnosis

The output:

```
path of original: letter_image14.jpg
path of spacing: spacing_letter_image14.jpg
Exception in Tkinter callback
```

I found out that there is no image path called “spacing_letter_image14.jpg”. This is why I got an error

My project: Handwriting recognition research project

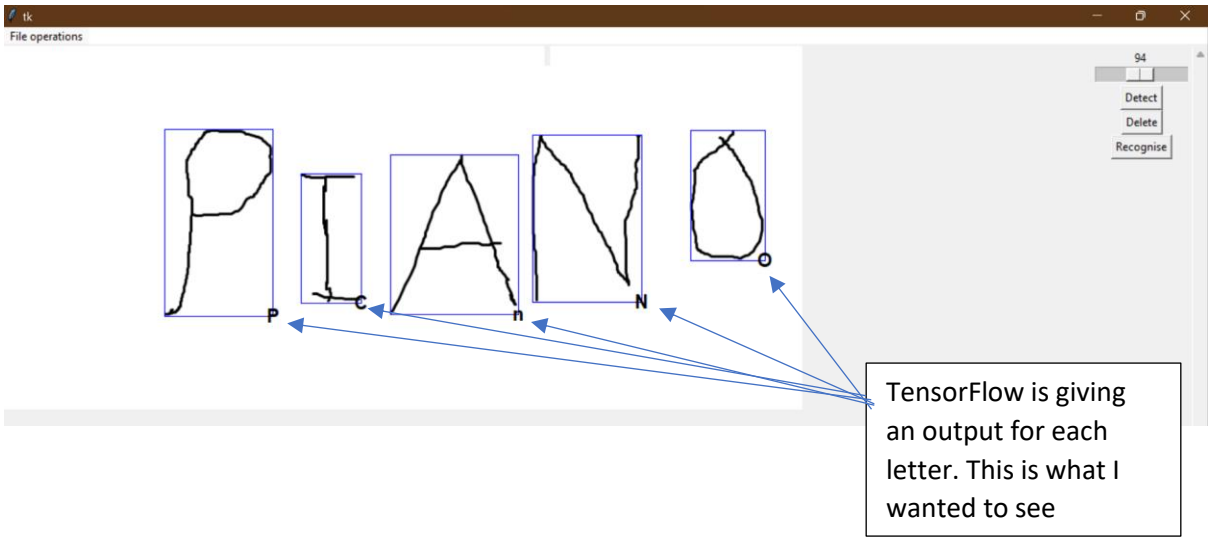
```
image_recog_list = []
for x in range(0, len(coor_info)):
    num = coor_info[x][1] # this is the unique identifier for each letter
    path = "letter_image{}.jpg".format(num) # path of letter
    print("path of original: {}".format(path))
    import spacing_for_letter
    img_letter = cv2.imread(path)  # image is loaded
    spacing_for_letter.main(img_letter, path) # spacing is applied on let
    path = "spacing_{}".format(path)
    print("path of spacing: {}".format(path))
    import thickness_of_letter_enlarged
    img_space = cv2.imread(path) # image is loaded
    thickness_of_letter_enlarged.main(img_space, pat
    path = "thick_{}".format(path)
    print("path of thick: {}".format(path))
    import TESTING_nn
    image_for_pred = TESTING_nn.process_letter(path) # image is processed
```

I realised that the path should be **"spaces_{}".format(path)** not **"spacing_{}".format(path)**

After:

```
path = "spaces_{}".format(path)
print("path of spaces: {}".format(path))
import thickness_of_letter_enlarged
```

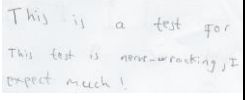
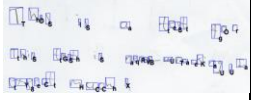
Output after amendments:

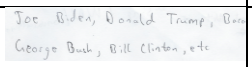


TensorFlow is giving an output for each letter. This is what I wanted to see

Using the sample image above, the TensorFlow results are not as impressive as I initially expected it to be. The program above gave a wrong answer for the letter image of I (as C) and A (as n).

Testing on my GUI so far (finalGUI.py):

Test	Input data/input code	Desired output	Actual output	Pass/ Fail	Explanation/Justification
I want to observe how my GUI detects letters in a		Ideally, I would like at least 40% of my handwri		Pass	I am pleasantly surprised that my neural network has recognised a lot of letters in this image. It managed to

handwritten image and what outputs I will get for recognition (Tensorflow)		tten data correctly detected and recognised by Tensorflow			recognise letters that form words like “GUI” and “test” which is very convincing to see
I want to observe how my GUI detects letters in a handwritten image and what outputs I will get for recognition (Tensorflow)		Ideally, I would like at least 40% of my handwritten data correctly detected and recognised by Tensorflow		Pass	Although I have said this test is a pass, I am a bit disappointed because the accuracy of the recognition algorithm was lower with this image than the previous image. However, my handwriting was also not clear enough and might have therefore influenced the poor performance.

Section 3.4.4) Making a recognition button (part 2)

Now that I know that my neural network works on my GUI input images, I will now implement **row_sorting.py** and **laying_out_words.py** into my GUI.

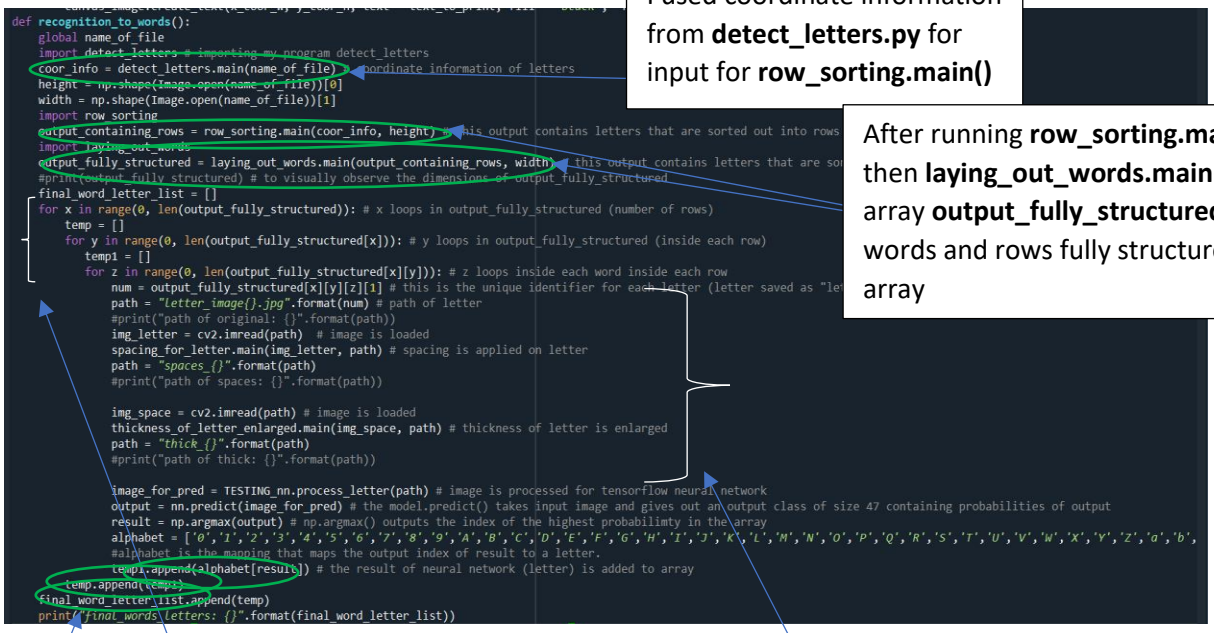
My plan is to output an array which contains rows/lines and array for word letters using **row_sorting.py** and **laying_out_words.py**, and then substitute the neural network output for the letter images so that words and sentences with real letters can be formed.

This is my code implementation for it:

My project: Handwriting recognition research project

I used coordinate information from **detect_letters.py** for input for **row_sorting.main()**

After running **row_sorting.main()** and then **laying_out_words.main()** I get an array **output_fully_structured**, with words and rows fully structured in the array



```
def recognition_to_words():
    global name_of_file
    import detect_letters # importing my program detect_letters
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    width = np.shape(Image.open(name_of_file))[1]
    import row_sorting
    output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into rows
    import laying_out_words
    output_fully_structured = laying_out_words.main(output_containing_rows, width) # this output contains letters that are so
    #print(output_fully_structured) # to visually observe the dimensions of output_fully_structured
    final_word_letter_list = []
    for x in range(0, len(output_fully_structured)): # x loops in output_fully_structured (number of rows)
        temp = []
        for y in range(0, len(output_fully_structured[x])): # y loops in output_fully_structured (inside each row)
            temp1 = []
            for z in range(0, len(output_fully_structured[x][y])): # z loops inside each word inside each row
                num = output_fully_structured[x][y][z][1] # this is the unique identifier for each letter (letter saved as "le
                path = "letter_image{}.jpg".format(num) # path of letter
                #print("path of original: {}".format(path))
                img_letter = cv2.imread(path)  # image is loaded
                spacing_for_letter.main(img_letter, path) # spacing is applied on letter
                path = "spaces_{}".format(path)
                #print("path of spaces: {}".format(path))

                img_space = cv2.imread(path) # image is loaded
                thickness_of_letter_enlarged.main(img_space, path) # thickness of letter is enlarged
                path = "thick_{}".format(path)
                #print("path of thick: {}".format(path))

                image_for_pred = TESTING_nn.process_letter(path) # image is processed for tensorflow neural network
                output = nn.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 containing probabilities of output
                result = np.argmax(output) # np.argmax() outputs the index of the highest probabilimty in the array
                alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b',
                #alphabet is the mapping that maps the output index of result to a letter.
                temp1.append(alphabet[result]) # the result of neural network (letter) is added to array
            temp.append(temp1)
        final_word_letter_list.append(temp)
    print("final_words\letters: {}".format(final_word_letter_list))
```

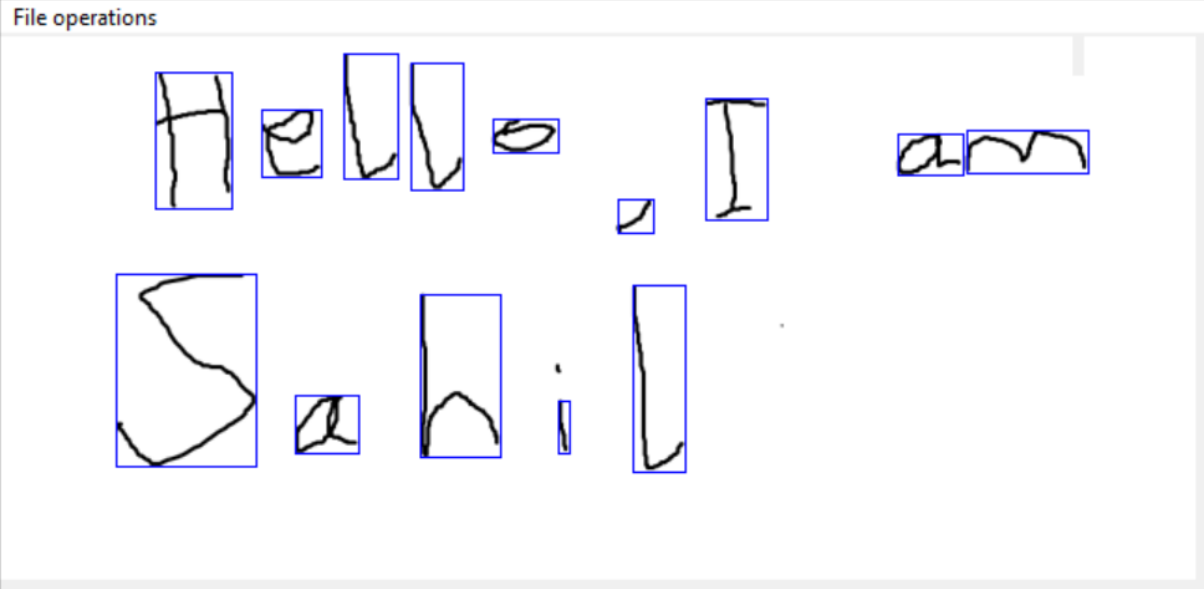
I have 3 for loops nested inside each other. The first one loops in each row, the second one loops in each row through each word, the third one loops through each letter in each word in each row.

This is the same code that I made in **recognition()** above, using the **num** value of each letter, I find the letter's path and make sure it is recognised by the neural network

After each letter is recognised, it is saved in **final_word_letter_list** which stores the recognised letters in the identical row, and word structure of **output_fully_structured**.

To test out my program, I will use this sample image:

My project: Handwriting recognition research project



The output of **final_word_letter_list** after pressing the “Recognition” button is:

```
final_words_letters: [[['L']]]
```

This is not what I wanted. I wanted letters recognised of each of the bounding boxes (14 bounding boxes in total) in the **final_word_letter_list.**

To diagnose the problem, I will put print statements in the for loops to see where the error lies.

```
print(output_fully_structured) # to visually obs                                    ed
final_word_letter_list = []
for x in range(0, len(output_fully_structured)):                                    of rows)
    temp = []
    for y in range(0, len(output_fully_structured[x])): # y loops in output_fully_structured (inside each row)
        temp1 = []
        for z in range(0, len(output_fully_structured[x][y])): # z loops inside each word inside each row
            print("x: {}, y: {}, z: {}".format(x, y, z))
            num = output_fully_structured[x][y][z][1] # this is the unique                 (letter saved as "let
            path = "letter_image{}.jpg".format(num) # path of letter
            #print("path of original: {}".format(path))
            img_letter = cv2.imread(path)  # image is loaded
            spacing_for_letter.main(img_letter, path) # spacing is applied
            path = "spaces_{}".format(path)
            #print("path of spaces: {}".format(path))

            img_space = cv2.imread(path) # image is loaded
            thickness_of_letter_enlarged.main(img_space, path) # thickness of letter is enlarged
            path = "thick_{}".format(path)
            #print("path of thick: {}".format(path))

            image_for_pred = TESTING_nn.process_letter(path) # image is processed for tensorflow neural network
            output = nn.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of si
            result = np.argmax(output) # np.argmax() outputs the index of the highest probabilimty in the array
            alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O
            #alphabet is the mapping that maps the output index of result to a letter.
            print("result of letter: {}".format(alphabet[result]))
            temp1.append(alphabet[result]) # the result of neural network (letter) is added to array
    temp.append(temp1)
```

Print statement to see if **output_fully_structured** is valid

Print statement to see what x,y,z points are passing in

Print statement to see if neural network is outputting as intended

Output:

My project: Handwriting recognition research project

```
x: 0, y: 0, z: 0
result of letter: E
x: 0, y: 0, z: 1
result of letter: 2
x: 0, y: 0, z: 2
result of letter: I
x: 0, y: 0, z: 3
result of letter: I
x: 0, y: 0, z: 4
result of letter: t
x: 0, y: 0, z: 5
result of letter: e
x: 0, y: 1, z: 0
result of letter: a
x: 0, y: 2, z: 0
result of letter: S
x: 0, y: 3, z: 0
result of letter: I
x: 1, y: 0, z: 0
result of letter: 4
x: 1, y: 1, z: 0
result of letter: X
x: 1, y: 2, z: 0
result of letter: 5
x: 1, y: 2, z: 1

2022-04-16 02:14:26.996437: I tensorflow/compiler/mlir/mlir_graph_optimization_pass
result of letter: L
x: 1, y: 3, z: 0
result of letter: O
final_words_letters: [[['O']]]
```

As I expected I am getting 14 outputs for neural network (as there are 14 letters in the input image). This means that something is wrong in my list appending

Amendment to my code:

```
            print("result of letter: {}".format(alphabet[result]))
            temp1.append(alphabet[result]) # the result of neural n
        temp.append(temp1)
    final_word_letter_list.append(temp)
print("final_words_letters: {}".format(final_word_letter_list))
```

Now that I know that the problem lies in my indentations of lists, I have made some tweaks to it.

Output:

```
result of letter: L
final_words_letters: [[['n', 'e', 'I', 'I', 'b',
'I'], ['I'], ['O'], ['N']], [['S'], ['a'], ['h',
'8'], ['L']]]
```

I have the output that I desired (14 letters in a list, partitioned out through rows and word sizes)

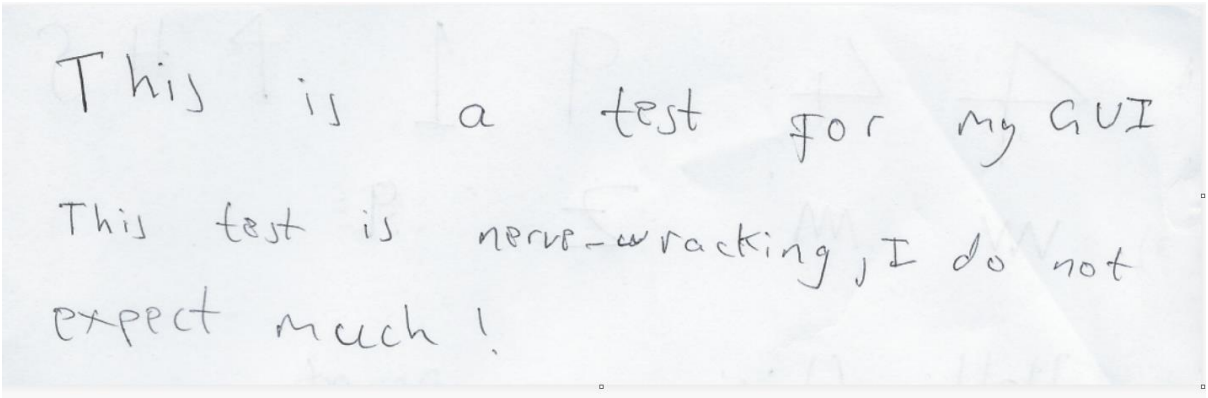
My code for **recognition_to_words()**:

My project: Handwriting recognition research project

```
def recognition_to_words():
    global name_of_file
    import detect_letters # importing my program detect_letters
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    width = np.shape(Image.open(name_of_file))[1]
    import row_sorting
    output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into rows
    import laying_out_words
    output_fully_structured = laying_out_words.main(output_containing_rows, width) # this output contains letters that are sorted out into rows and words in each line
    print(output_fully_structured) # to visually observe the dimensions of output_fully_structured
    final_word_letter_list = []
    for x in range(0, len(output_fully_structured)): # x loops in output_fully_structured (number of rows)
        temp = []
        for y in range(0, len(output_fully_structured[x])): # y loops in output_fully_structured (inside each row)
            temp1 = []
            for z in range(0, len(output_fully_structured[x][y])): # z loops inside each word inside each row
                print("x: {}, y: {}, z: {}".format(x, y, z))
                num = output_fully_structured[x][y][z][1] # this is the unique identifier for each letter (letter saved as "letter_image" + num)
                path = "letter_image{}.jpg".format(num) # path of letter
                #print("path of original: {}".format(path))
                img_letter = cv2.imread(path)  # image is loaded
                spacing_for_letter.main(img_letter, path) # spacing is applied on letter
                path = "spaces_{}".format(path)
                #print("path of spaces: {}".format(path))

                img_space = cv2.imread(path) # image is loaded
                thickness_of_letter_enlarged.main(img_space, path) # thickness of letter is enlarged
                path = "thick_{}".format(path)
                #print("path of thick: {}".format(path))

                image_for_pred = TESTING_nn.process_letter(path) # image is processed for tensorflow neural network
                output = nn.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 containing probabilities of output
                result = np.argmax(output) # np.argmax() outputs the index of the highest probabilimty in the array
                alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','d','e','f','g','h','n','q','r','t']
                #alphabet is the mapping that maps the output index of result to a letter.
                print("result of letter: {}".format(alphabet[result]))
                temp1.append(alphabet[result]) # the result of neural network (letter) is added to array
            temp.append(temp1)
        final_word_letter_list.append(temp)
    print("final_words_letters: {}".format(final_word_letter_list))
    rows = []
    for x in range(0, len(final_word_letter_list)):
        in_row = ""
        for y in range(0, len(final_word_letter_list[x])):
            str1 = ""
            for z in range(0, len(final_word_letter_list[x][y])):
                str1 += final_word_letter_list[x][y][z]
            in_row += " " + str1
        rows.append(in_row)
    print(rows)
recognition_button = Button(root, text = "Recognise", command = recognition_to_words)
recognition_button.pack()
```

Since my function works in outputting letters into rows and words (in each row), I made a array **rows** which can concatenate multiple letters to form words (and gaps between words). Each element of **rows** is a row of string

Sample image (that I used in test):



This is the output that I get:

```
['ThDS IS a teSt gOr M7gUI', ' thStGShS aqRVBUTaeKaUUaUOMOt', ' tY8eCt HCChX']
```

My program models the input image data, into 3 rows of sentences that have the roughly the same size for each word. However, the words don't really make any sense

Section 3.4.5) Making a new window for output:

My project: Handwriting recognition research project

Now that my recognition is working, I need to work on making a new window, that opens as soon as the recognition button is pressed.

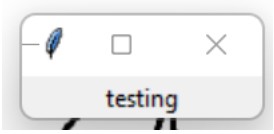
Since I was not sure, how to make a new window, I got help from [Open a new Window with a button in Python-Tkinter - GeeksforGeeks]

This is my code based on GeeksforGeeks.com

```
def new_window():
    new_root = Toplevel(root) # makes a new window
    a = Label(new_root, text = "testing")
    a.pack()
new_window = Button(root, text = "New window", command = new_window)
new_window.pack()
root.mainloop()
```

I will display a label just as a test

This is the output when I clicked “New window”:



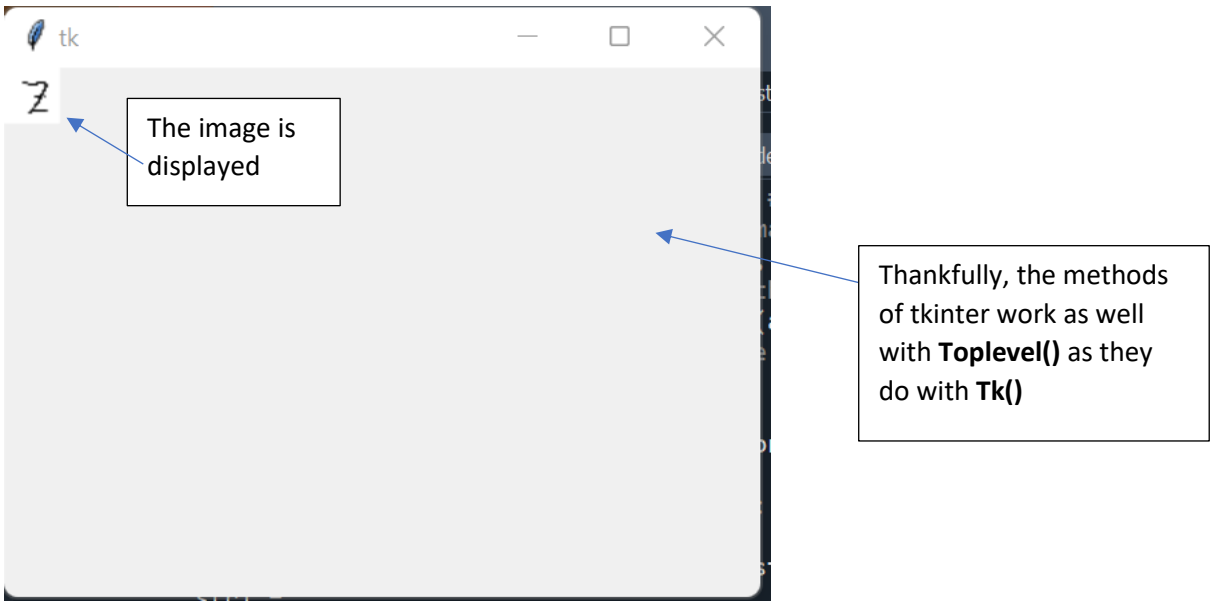
Next, I need to test this new window by using PIL ImageTk, to see if it displays images, or not. This is my code adaptation for the test:

```
def new_window():
    new_root = Toplevel(root) # makes a new window
    canvas_image2 = Canvas(new_root) # canvas widget is defined where I will place my input ima
    canvas_image2.config(highlightthickness=0)
    canvas_image2.pack(side=LEFT, expand=YES, fill=BOTH) # .pack() allocates a coordinate space
    imageOpen = Image.open("d.png") # opening input image
    image1 = ImageTk.PhotoImage(imageOpen) # PhotoImage object that will be used to output imag
    image_plot = canvas_image2.create_image(0,0, anchor="nw",image=image1) # .create_image() di
    canvas_image2.image = image1
```

This is the same code that I used to make my Tkinter canvas (using **root**) however, I changed the root of the canvas to **new_root**

My project: Handwriting recognition research project

This is the output to my code:



Before I work further with Tkinter Toplevel(), I want to make sure I can crop each row of the input image using slicing (so I can display it on the new window).

This is my code to test slicing of images:

```
import cv2
import detect_letters
import row_sorting
import laying_out_words
from PIL import Image
import numpy as np
a = cv2.imread("3_rows_roughly.png")
height = np.shape(Image.open("3_rows_roughly.png"))[0]
width = np.shape(Image.open("3_rows_roughly.png"))[1]
coor_info = detect_letters.main("3_rows_roughly.png")
output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into rows
row_sorted = []
roi = 0
for x in range(0, len(output_containing_rows)):
    roi += 1
    row_sorted = laying_out_words.bubbleSort_sort_column(output_containing_rows[x]) # within each row, the list is sorted by x coordinates
    temp = a[int(round(row_sorted[0][0][1])):int(round(row_sorted[-1][0][1][1])), int(round(row_sorted[0][0][0][0])): int(round(row_sorted[-1][0][1][0]))]
    cv2.imwrite("r{}.png".format(roi), temp)
cv2.waitKey()
```

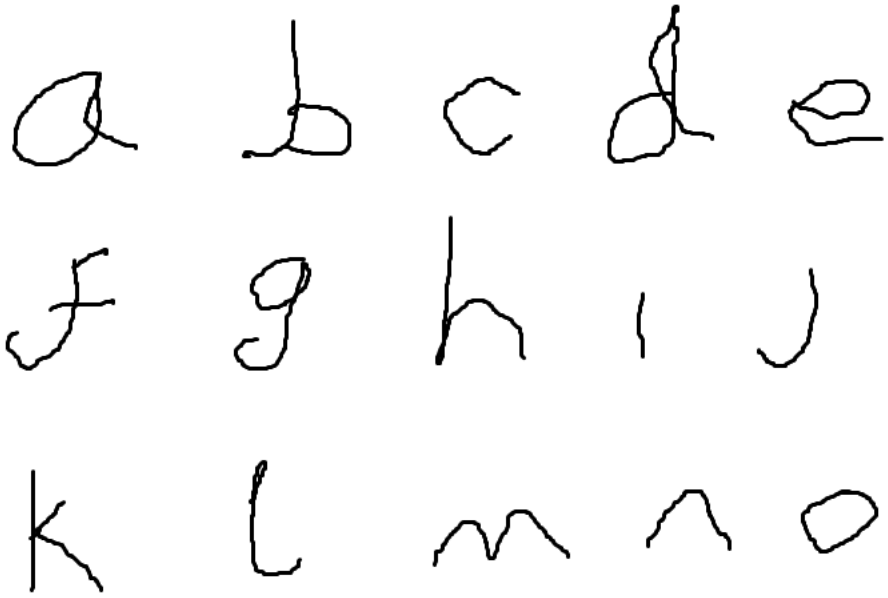
I use **detect_letters.py** to assign to **coor_info**. I then use it inside **row_sorting.py**, the output of which is an array with lists of rows

row_sorted is an array representing the current row (element of index **x**) in **output_containing_rows**

I use slicing to crop out one row at a time. I use the first element coordinates **row_sorted** and last element coordinates as slicing parameters

My project: Handwriting recognition research project

I tested my code on this sample image:



The output of my program:

Row 1:

Row 2:

Row 3:

Looking at my outputs above, I can conclude that my slicing algorithm works and I can implement it in my Tkinter program.

My project: Handwriting recognition research project

This is my Tkinter implementation (in my **new_window()**):

```
def new_window():
    global name_of_file
    new_root = Toplevel(root) # makes a new window
    canvas_image2 = Canvas(new_root) # canvas widget is defined where I will place my input image
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    width = np.shape(Image.open(name_of_file))[1]
    coor_info = detect_letters.main(name_of_file)
    output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into
    row_sorted = []
    row = 0
    img_file = cv2.imread(name_of_file)
    for x in range(0, len(output_containing_rows)):
        row += 1
        row_sorted = laying_out_words.bubbleSort_sort_column(output_containing_rows[x])# within each row, the list is so
        temp = img_file[int(round(row_sorted[0][0][0][1])):int(round(row_sorted[-1][0][1][1])), int(round(row_sorted[0][
        cv2.imwrite("row{}.png".format(row), temp)
    save_ImageTk_instance = []
    save_pos_previous_image = 0
    for x in range(0, len(output_containing_rows)):
        save_pos_previous_image += 50
        print("in here")
        filename = "row{}.png".format(x+1)
        file_open = Image.open(filename)
        file_open = file_open.resize((700,60))
        photo = ImageTk.PhotoImage(file_open)
        save_ImageTk_instance.append(photo)
        canvas_image2.create_image(0,save_pos_previous_image,anchor=CENTER, image = save_ImageTk_instance[-1])
```

When I load up the row image for displaying, I resize them to (700, 60) to maintain consistency when displaying the image, otherwise they look uneven

This part is the same as the code above (its used to crop and save the rows into images)

The coordinates of the rows will have the same x -coordinate (because they have the same horizontal position) but different y-coordinates (increment by 50 each time)

I saved the current instance of **ImageTk.PhotoImage()** in **save_ImageTk_instance** because according to python - Why does Tkinter image not show up if created in a function? - Stack Overflow it helps retain the references to **ImageTk.PhotoImage()**

The Output:

My project: Handwriting recognition research project

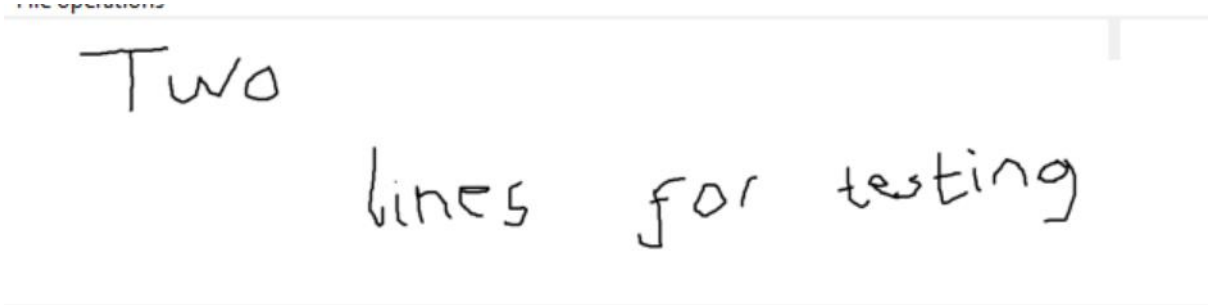
I need to increase the spacing between the 3 rows because right now all 3 of the images are on top of each other and hence some of their details are cut

I incremented the y-coordinate gap between each adjacent image from 50 to 70. This is the new output:

Each of the images can clearly be seen now.

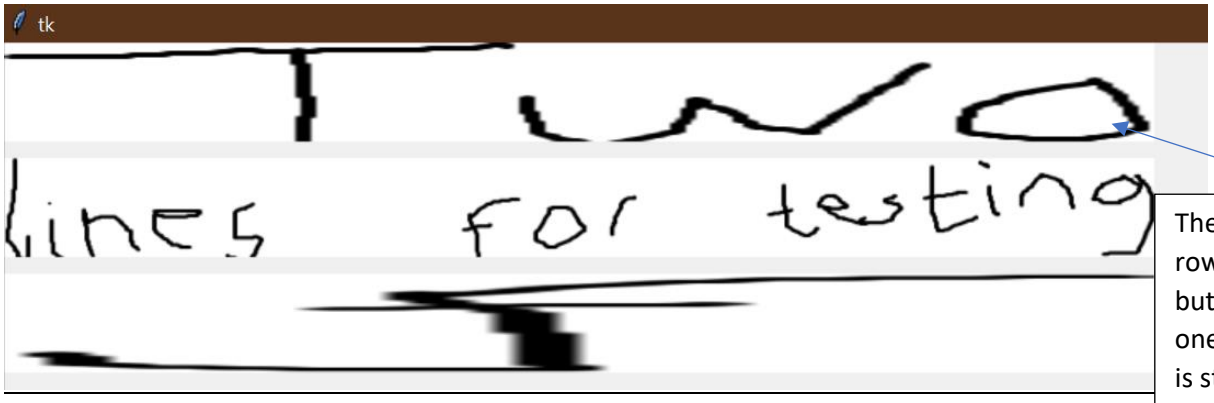
I will use another sample image to make sure that the scaling of each row is even amongst all input images.

This is my sample:



My project: Handwriting recognition research project

This is the output:



The second row scales well but the first one doesn't (it is stretched too far for its aspect ratio)

I will not make changes to my program above to facilitate smaller images because the purpose of these rows aren't to look aesthetic for the program but as a check for the user (if they want add something to the final output file) so I will not worry about scenarios like row 1 in the output above.

Now that I have successfully displayed my row images, I will now add entry boxes underneath each row image where ultimately, the output of each line will be written and the user can amend it.

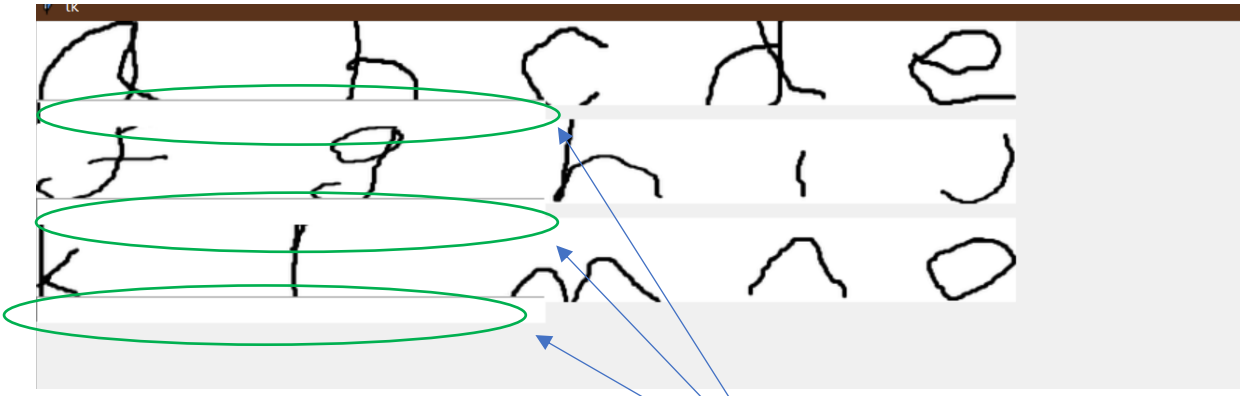
```
def new_window():
    global name_of_file
    new_root = Toplevel(root) # makes a new window
    canvas_image2 = Canvas(new_root) # canvas widget is defined where I will place my input image
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    width = np.shape(Image.open(name_of_file))[1]
    coor_info = detect_letters.main(name_of_file)
    output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into rows
    row_sorted = []
    row = 0
    img_file = cv2.imread(name_of_file)
    for x in range(0, len(output_containing_rows)):
        row += 1
        row_sorted = laying_out_words.bubbleSort_sort_column(output_containing_rows[x])# within each row, the list is sorted by x coordinates
        temp = img_file[int(round(row_sorted[0][0][0][1])):int(round(row_sorted[-1][0][1][1])), int(round(row_sorted[0][0][0][0])): int(round(row_sorted[-1][0][1][0]))
        cv2.imwrite("row{}.png".format(row), temp)
    save_ImageTk_instance = []
    save_pos_previous_image = 0
    for x in range(0, len(output_containing_rows)):
        save_pos_previous_image += 70
        print("in here")
        filename = "row{}.png".format(x+1)
        file_open = Image.open(filename)
        file_open = file_open.resize((700,60), Image.ANTIALIAS)
        a = Entry(canvas_image2, width = 60)
        canvas_image2.create_window(0, save_pos_previous_image + 35 , anchor = "w", window = a)
        photo = ImageTk.PhotoImage(file_open)
        save_ImageTk_instance.append(photo)
        canvas_image2.create_image(0,save_pos_previous_image,anchor="w", image = save_ImageTk_instance[-1])
    vertical_scrollbar = Scrollbar(new_root, orient = "vertical", command = canvas_image.yview) # defining vertical scrollba
    vertical_scrollbar.pack(side = RIGHT, fill = Y)
    horizontal_scrollbar = Scrollbar(new_root, orient = "horizontal", command = canvas_image.xview) # defining horizontal s
    horizontal_scrollbar.pack(side = BOTTOM, fill = X) # placement of this scrollbar is at the bottom
    horizontal_scrollbar.config(command=canvas_image2.xview)
    vertical_scrollbar.config(command=canvas_image2.yview)
    canvas_image2.configure(yscrollcommand = vertical_scrollbar.set, xscrollcommand = horizontal_scrollbar.set) # this is wh
    canvas_image2.configure(scrollregion=canvas_image2.bbox("all"))
    canvas_image2.config(highlightthickness=0)
    canvas_image2.pack(side=LEFT, expand=YES, fill=BOTH) # .pack() allocates a coordinate space for canvas_image
    new_root.mainloop()
new_window = Button(root, text = "New window", command = new_window)
new_window.pack()
root.mainloop()
```

I have also added code for scrollbars for canvas (which is the same code I wrote for canvas in the main **root** window)

Entry() is the Tkinter method for input boxes. I will also need to use **canvas_image2.create_window()** and assign **Entry()** as my window parameter, because I will need to use the canvas positioning for input boxes (as it uses coordinate system and the row image is also a canvas object and since I need the input box directly below the image, I will have to use the same positioning system as the row image)

My project: Handwriting recognition research project

The output:



The entry boxes correspond to each of the row images

I will now call my **recognition_to_words()** function I made, and use it as pre written text in my input boxes for each row.

```
def new_window():
    global name_of_file
    new_root = Toplevel(root) # makes a new window
    canvas_image2 = Canvas(new_root) # canvas widget is defined where I will place my input image
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    width = np.shape(Image.open(name_of_file))[1]
    coor_info = detect_letters.main(name_of_file)
    output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into rows
    row_sorted = []
    row = 0
    img_file = cv2.imread(name_of_file)
    string_words = recognition_to_words()
    for x in range(0, len(output_containing_rows)):
        row += 1
        row_sorted = laying_out_words.bubbleSort_sort_column(output_containin
        temp = img_file[int(round(row_sorted[0][0][0][1])):int(round(row_sort
        cv2.imwrite("row{}.png".format(row), temp)
    save_ImageTk_instance = []
    save_pos_previous_image = 0
    entry_get = []
    for x in range(0, len(output_containing_rows)):
        save_pos_previous_image += 70
        print("in here")
        filename = "row{}.png".format(x+1)
        file_open = Image.open(filename)
        file_open = file_open.resize((700,60), Image.ANTIALIAS)
        a = Entry(canvas_image2, width = 60)
        a.insert(0, string_words[x])
        canvas_image2.create_window(0, save_pos_previous_image + 35 , anchor = "w", window =
        entry_get.append(a)
        photo = ImageTk.PhotoImage(file_open)
        save_ImageTk_instance.append(photo)
        canvas_image2.create_image(0,save_pos_previous_image,anchor="w", image = save_ImageTk
    def writeDoc():
        #global entry_get
        final_write_list = []
        openFile = open("generated_output.txt", "w")
        for x in range(0, len(entry_get)):
            openFile.write("\n" + entry_get[x].get())
        openFile.close()
        from tkinter import messagebox
        messagebox.showinfo("showinfo", "File saved as generated_output.txt")
    new_window = Button(root, text = "New window", command = new_window)
    new_window.pack()
```

I saved the output of **recognition_to_words()** to **string_words**

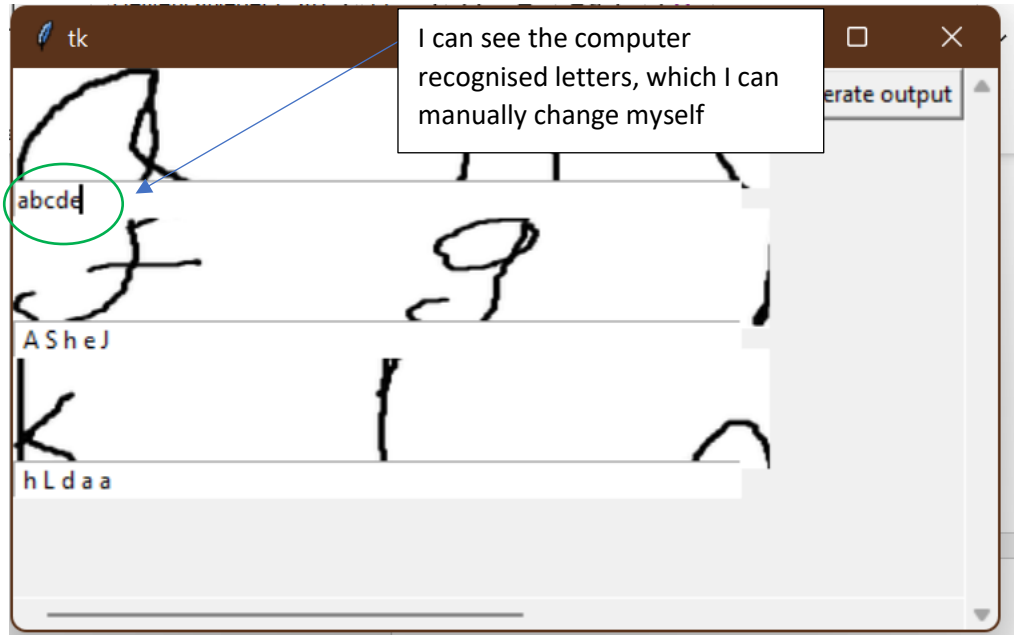
Since each element in **string_words** represents a row, I will enter this pre written dat

Since I can't individually assign every **Entry()** method call (because) I don't know how many rows each image will have, I have saved each **Entry()** object in a list **entry_get**

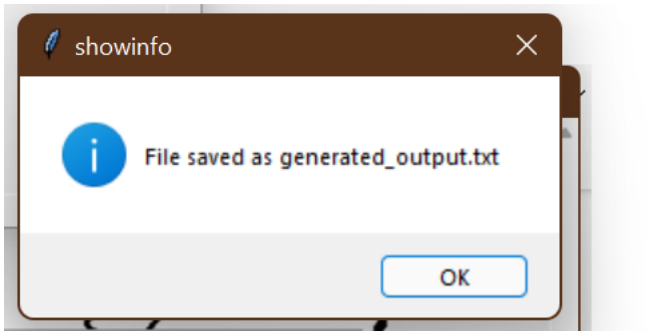
I write the text in the file "generated_output.txt" and later output a message sayiny "File saved successfully" using tkinter.messagebox()

Output of the code above:

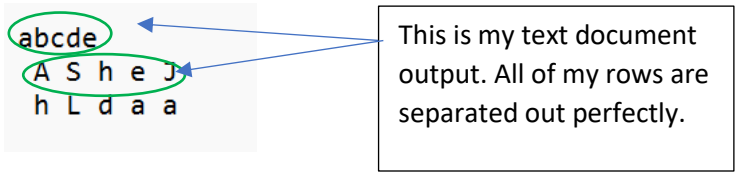
In the new Topleyer() window, I get the this output:



When I click the “generate output” button, I get the message:



Inside file **generated_output.txt**:



Section 3.4.6) Writing clear labels for each window:

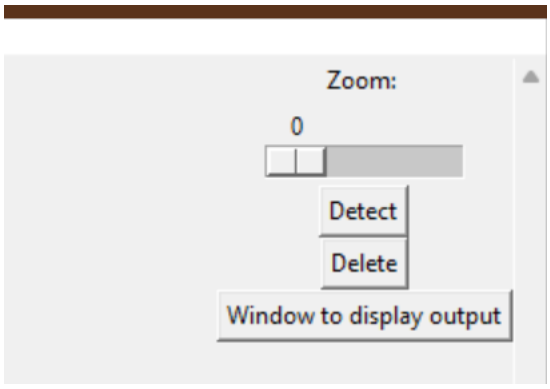
My project: Handwriting recognition research project

This is just a very small change that I made to my program.

```
scale_factor = 1
def zoom(event):
    global image
    global scale_factor
    global counter
    scale_factor = slider1.get() # this .get() method receives the input that the user gives for Scale()
    width = int(np.shape(imageOpen)[1]*scale_factor/100) # I change the width of the original image using scale factor
    height = int(np.shape(imageOpen)[0]*scale_factor/100) # I change the height of the original image using scale factor
    if width == 0:
        width = 1
    if height == 0:
        height = 1
    imageResize = imageOpen.resize((width, height)) # resizing image based on new width and height
    image = ImageTk.PhotoImage(image = imageResize)
    canvas_image.itemconfig(image_plot, image = image) # .itemconfig is
    canvas_image.delete("rect")
    resize_bounding_with_zoom()
    if counter == 1:
        recognition()
label = Label(root, text = "Zoom: ")
label.pack()
slider1 = Scale(root, from_ = 0, to = 200, orient = HORIZONTAL, command = zoom) # this is the slider that is used for zoo
slider1.pack()
coor_info = []
```

I added a label here to illustrate the slider is for zooming

Output:

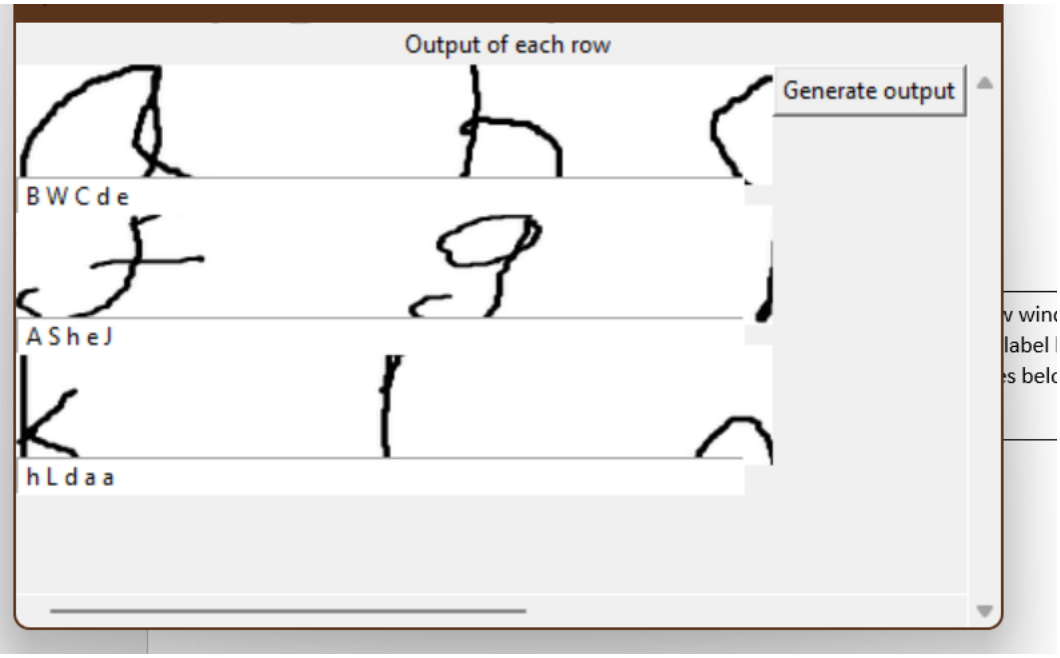



```
new_root = Toplevel(root) # makes a new window
label = Label(new_root, text = "Output of each row")
label.pack()
canvas_image2 = Canvas(new_root) # canvas widget is defined where I
coor_info = detect_letters.main(name_of_file) # coordinate informati
height = np.shape(Image.open(name_of_file))[0]
width = np.shape(Image.open(name_of_file))[1]
```

In the new window for output, I written a label here that indicates the images below refer to rows of input

Output:

My project: Handwriting recognition research project



Final code for Graphical User Interface:

My project: Handwriting recognition research project

```
"""
from tkinter import *
from PIL import Image, ImageTk
import numpy as np
import cv2
import tensorflow as tf
import spacing_for_letter
import thickness_of_letter_enlarged
import TESTING_nn
import row_sorting
import laying_out_words
import detect_letters
nn = tf.keras.models.load_model("final_CNN") # loading my CNN model
name_of_file = "3_rows_roughly.png" # I will use this when I need to call functions from external python files

def open_file():
    global name_of_file
    global image
    global imageOpen
    filetypes = (
        ("PNG file", '*.png'),
        ("JPEG/JPG file", '*.jpeg *.jpg') # these are the only types of files acceptable when using filedialog.askopenfile()
        )
    from tkinter import filedialog
    name_of_file = filedialog.askopenfilename( # this is a tkinter method that opens a window and allows the user to manually search their directories to open a file
        title = "Open a file",
        initialdir = "/",
        filetypes = filetypes # I have defined this above, PNG and JPEG/JPG are the only filetypes allowed
        )
    imageOpen = Image.open(name_of_file) # opening the file path that the user has input
    image = ImageTk.PhotoImage(image = imageOpen)
    canvas_image.itemconfig(image_plot, image = image) # displays the image onto canvas_image
    canvas_image.delete("rect")

root = Tk() #defining the main window of the program
menu1 = Menu(root) # the Menu() defines a menu for me in the window
menu_dropdown = Menu(menu1, tearoff = 0)
menu_dropdown.add_command(label = "Open", command = open_file) # this is the actual button that will allow the user to input images
menu_dropdown.add_separator()
menu1.add_cascade(label = "File operations", menu = menu_dropdown) # this is the tab the user has to click on to get to the "Open" button
root.config(menu = menu1) # stating that the main window will have a menubar
canvas_image = Canvas(root) # canvas widget is defined where I will place my input image
canvas_image.config(highlightthickness=0)
canvas_image.pack(side=LEFT, expand=YES, fill=BOTH) # .pack() allocates a coordinate space for canvas_image
imageOpen = Image.open(name_of_file) # opening input image
image = ImageTk.PhotoImage(imageOpen) # PhotoImage object that will be used to output image in canvas_image
image_plot = canvas_image.create_image(0,0, anchor="nw",image=image) # .create_image() displayes my input image on the canvas
vertical_scrollbar = Scrollbar(root, orient = "vertical", command = canvas_image.yview) # defining vertical scrollbar
vertical_scrollbar.pack(side = RIGHT, fill = Y)
horizontal_scrollbar = Scrollbar(root, orient = "horizontal", command = canvas_image.xview) # defining horizontal scrollbar
horizontal_scrollbar.pack(side = BOTTOM, fill = X) # placement of this scrollbar is at the bottom
horizontal_scrollbar.config(command=canvas_image.xview)
vertical_scrollbar.config(command=canvas_image.yview)
canvas_image.configure(yscrollcommand = vertical_scrollbar.set, xscrollcommand = horizontal_scrollbar.set) # this is where I assign the scrollbar command onto canvas_image specifically
canvas_image.configure(scrollregion=canvas_image.bbox("all"))
canvas_image.config(scrollregion=(0,0,np.shape(imageOpen)[1], np.shape(imageOpen)[0]))
label = Label(canvas_image)
label.pack()
scale_factor = 1
def zoom(event):
    global image
    global scale_factor
    global counter
    scale_factor = slider1.get() # this .get() method receives the input that the user gives for Scale()
```

My project: Handwriting recognition research project

```
    global counter
    scale_factor = slider1.get() # this .get() method receives the input that the user gives for Scale()
    width = int(np.shape(imageOpen)[1]*scale_factor/100) # I change the width of the original image using scale factor
    height = int(np.shape(imageOpen)[0]*scale_factor/100) # I change the height of the original image using scale factor
    if width == 0:
        width = 1
    if height == 0:
        height = 1
    imageResize = imageOpen.resize((width, height)) # resizing image based on new width and height
    image = ImageTk.PhotoImage(image = imageResize)
    canvas_image.itemconfig(image_plot, image = image) # .itemconfig is used to update imageplot which deals with displaying images (in above code)
    canvas_image.delete("rect")
    resize_bounding_with_zoom()
    if counter == 1:
        recognition()
label = Label(root, text = "Zoom: ")
label.pack()
slider1 = Scale(root, from_ = 0, to = 200, orient = HORIZONTAL, command = zoom) # this is the slider that is used for zooming, with minimum value = 0, maximum value = 200
slider1.pack()
coor_info = []
def detect():
    global coor_info # I have used coor_info as a global variable so I can use it throughout the program
    import detect_letters # importing my program detect_letters
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    import row_sorting
    output = row_sorting.main(coor_info, height)
    #print("output: " + str(output))
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0][0]*scale_factor/100)) # x coordinate of bounding box
        y_coor = int(round(coor_info[x][0][0][1]*scale_factor/100)) # y coordinate of bounding box
        x_coor_w = int(round(coor_info[x][0][1][0]*scale_factor/100)) # x + w coordinate of bounding box
        y_coor_h = int(round(coor_info[x][0][1][1]*scale_factor/100)) # # y + h coordinate of bounding box
        #print(x_coor, y_coor, x_coor_w, y_coor_h)
        canvas_image.create_rectangle(x_coor, y_coor, x_coor_w, y_coor_h, outline = "blue", tags = "rect") # creates bounding box
        text_to_print = "x:{}, y:{}, num: {}".format(x_coor_w, y_coor_h, coor_info[x][1]) # text for each bounding box showing its num (from detect letters) and current coordinates
        canvas_image.create_text(x_coor_w, y_coor_h, text = text_to_print, fill = "black", tags = "rect") # outputs text
def resize_bounding_with_zoom():
    global coor_info # I have used coor_info from detect as a global variable so I can use it throughout the program
    global scale_factor
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0][0]*scale_factor/100)) # x coordinate of bounding box after scale factor
        y_coor = int(round(coor_info[x][0][0][1]*scale_factor/100)) # y coordinate of bounding box after scale factor
        x_coor_w = int(round(coor_info[x][0][1][0]*scale_factor/100)) # x + w coordinate of bounding box after scale factor
        y_coor_h = int(round(coor_info[x][0][1][1]*scale_factor/100)) # # y + h coordinate of bounding box after scale factor
        #print(x_coor, y_coor, x_coor_w, y_coor_h)
        canvas_image.create_rectangle(x_coor, y_coor, x_coor_w, y_coor_h, outline = "blue", tags = "rect")
        text_to_print = "x:{}, y:{}, num: {}".format(x_coor_w, y_coor_h, coor_info[x][1])
        canvas_image.create_text(x_coor_w, y_coor_h, text = text_to_print, fill = "black",  tags = "rect")
detect_button = Button(root, text = "Detect", command = detect)
detect_button.pack()
def delete_canvas_rectangles():
    canvas_image.delete("rect")
delete_button = Button(root, text = "Delete", command = delete_canvas_rectangles)
delete_button.pack()
counter = 0
def recognition():
    global counter
    counter = 1
    global nn
    global scale_factor
    image_recog_list = []
    for x in range(0, len(coor_info)):
        num = coor_info[x][1] # this is the unique identifier for each letter (letter saved as "letter_image" + num)
        path = "letter_image{}.jpg".format(num) # path of letter
        #print("path of original: {}".format(path))

        img_letter = cv2.imread(path)  # image is loaded
        spacing_for_letter.main(img_letter, path) # spacing is applied on letter
        path = "spaces/1" format(path)
```

My project: Handwriting recognition research project

```
        img_space = cv2.imread(path) # image is loaded
        thickness_of_letter_enlarged.main(img_space, path) # thickness of letter is enlarged
        path = "thick_{}".format(path)
        #print("path of thick: {}".format(path))

        image_for_pred = TESTING_nn.process_letter(path) # image is processed for tensorflow neural network
        output = nn.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 containing probabilities of output
        result = np.argmax(output) # np.argmax() outputs the index of the highest probabilimty in the array
        alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','d','e','f','g','h','n','q','r','t']
        #alphabet is the mapping that maps the output index of result to a letter.
        image_recog_list.append(alphabet[result]) # the result of neural network (letter) is added to array
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0][0]*scale_factor/100)) # x coordinate of bounding box after scale factor
        y_coor = int(round(coor_info[x][0][0][1]*scale_factor/100)) # y coordinate of bounding box after scale factor
        x_coor_w = int(round(coor_info[x][0][1][0]*scale_factor/100)) # x + w coordinate of bounding box after scale factor
        y_coor_h = int(round(coor_info[x][0][1][1]*scale_factor/100)) # # y + h coordinate of bounding box after scale factor
        #print(x_coor, y_coor, x_coor_w, y_coor_h)
        canvas_image.create_rectangle(x_coor, y_coor, x_coor_w, y_coor_h, outline = "blue", tags = "rect")
        text_to_print = "{}".format(image_recog_list[x]) # result of recognition is shown
        canvas_image.create_text(x_coor_w, y_coor_h, text = text_to_print, fill = "black",  font=('Helvetica 15 bold') , tags = "rect")
def recognition_to_words():
    global name_of_file
    import detect_letters # importing my program detect_letters
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    width = np.shape(Image.open(name_of_file))[1]
    import row_sorting
    output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into rows
    import laying_out_words
    output_fully_structured = laying_out_words.main(output_containing_rows, width) # this output contains letters that are sorted out into rows and words in each line
    print(output_fully_structured) # to visually observe the dimensions of output_fully_structured
    final_word_letter_list = []
    for x in range(0, len(output_fully_structured)): # x loops in output_fully_structured (number of rows)
        temp = []
        for y in range(0, len(output_fully_structured[x])): # y loops in output_fully_structured (inside each row)
            temp1 = []
            for z in range(0, len(output_fully_structured[x][y])): # z loops inside each word inside each row
                print("x: {}, y: {}, z: {}".format(x, y, z))
                num = output_fully_structured[x][y][z][1] # this is the unique identifier for each letter (letter saved as "letter_image" + num)
                path = "letter_image{}.jpg".format(num) # path of letter
                #print("path of original: {}".format(path))
                img_letter = cv2.imread(path)  # image is loaded
                spacing_for_letter.main(img_letter, path) # spacing is applied on letter
                path = "spaces_{}".format(path)
                #print("path of spaces: {}".format(path))

                img_space = cv2.imread(path) # image is loaded
                thickness_of_letter_enlarged.main(img_space, path) # thickness of letter is enlarged
                path = "thick_{}".format(path)
                #print("path of thick: {}".format(path))

                image_for_pred = TESTING_nn.process_letter(path) # image is processed for tensorflow neural network
                output = nn.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 containing probabilities of output
                result = np.argmax(output) # np.argmax() outputs the index of the highest probabilimty in the array
                alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','d','e','f','g','h','n','q','r','t']
                #alphabet is the mapping that maps the output index of result to a letter.
                print("result of letter: {}".format(alphabet[result]))
                temp1.append(alphabet[result]) # the result of neural network (letter) is added to array
            temp.append(temp1)
        final_word_letter_list.append(temp)
    print("final_words_letters: {}".format(final_word_letter_list))
    rows = []
    for x in range(0, len(final_word_letter_list)):
        in_row = ""
        for y in range(0, len(final_word_letter_list[x])):
            str1 = ""
            for z in range(0, len(final_word_letter_list[x][y])):
                str1 += final_word_letter_list[x][y][z]
            in_row += " " + str1
```

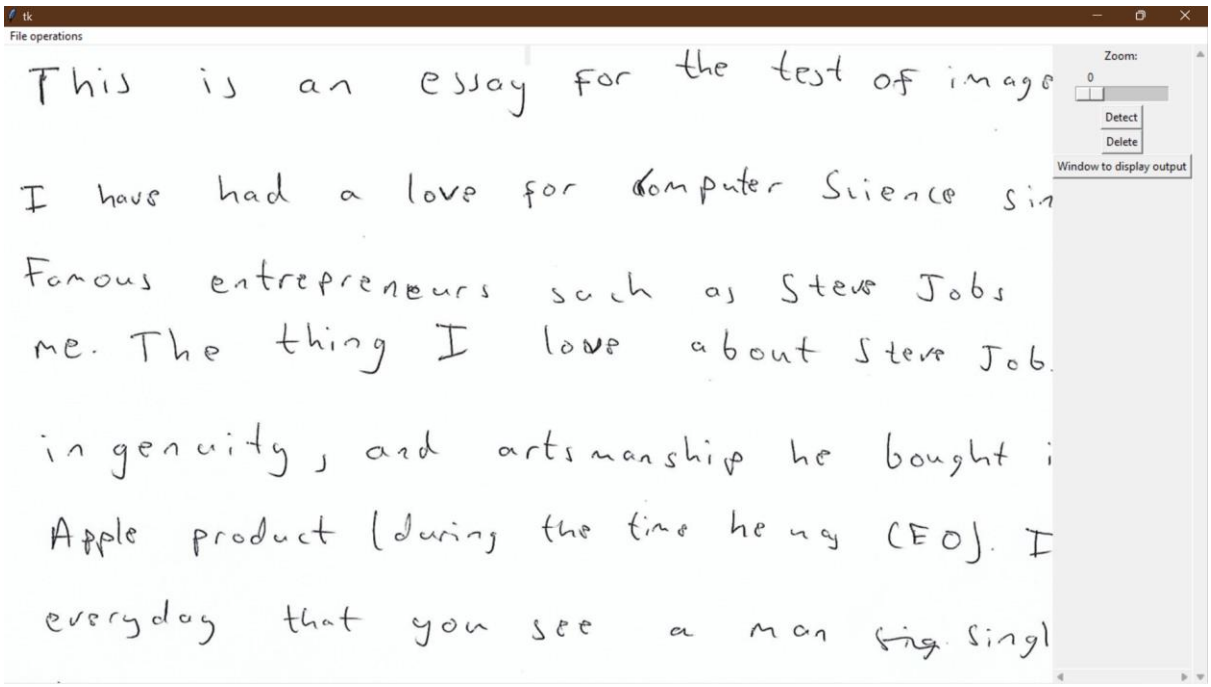
custom (Python 3.9.1)

```
def new_window():
    global name_of_file
    new_root = Toplevel(root) # makes a new window
    label = Label(new_root, text = "Output of each row")
    label.pack()
    canvas_image2 = Canvas(new_root) # canvas widget is defined where I will place my input image
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    width = np.shape(Image.open(name_of_file))[1]
    coor_info = detect_letters.main(name_of_file)
    output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into rows
    row_sorted = []
    row = 0
    img_file = cv2.imread(name_of_file)
    string_words = recognition_to_words()
    for x in range(0, len(output_containing_rows)):
        row += 1
        row_sorted = laying_out_words.bubbleSort_sort_column(output_containing_rows[x])# within each row, the list is sorted by x coordinates
        temp = img_file[int(round(row_sorted[0][0][0][1])):int(round(row_sorted[-1][0][1][1])), int(round(row_sorted[0][0][0][0])): int(round(row_sorted[-1][0][1][0]))]
        cv2.imwrite("row{}.png".format(row), temp)
    save_ImageTk_instance = []
    save_pos_previous_image = 0
    entry_get = []
    for x in range(0, len(output_containing_rows)):
        save_pos_previous_image += 70
        print("in here")
        filename = "row{}.png".format(x+1)
        file_open = Image.open(filename)
        file_open = file_open.resize((700,60), Image.ANTIALIAS)
        a = Entry(canvas_image2, width = 60)
        a.insert(0, string_words[x])
        canvas_image2.create_window(0, save_pos_previous_image + 35 , anchor = "w", window = a)
        entry_get.append(a)
        photo = ImageTk.PhotoImage(file_open)
        save_ImageTk_instance.append(photo)
        canvas_image2.create_image(0,save_pos_previous_image,anchor="w", image = save_ImageTk_instance[-1])
    def writeDoc():
        #global entry_get
        final_write_list = []
        openFile = open("generated_output.txt", "w")
        for x in range(0, len(entry_get)):
            openFile.write("\n" + entry_get[x].get())
        openFile.close()
        from tkinter import messagebox
        messagebox.showinfo("showinfo", "File saved as generated_output.txt")
    vertical_scrollbar = Scrollbar(new_root, orient = "vertical", command = canvas_image.yview) # defining vertical scrollbar
    vertical_scrollbar.pack(side = RIGHT, fill = Y)
    horizontal_scrollbar = Scrollbar(new_root, orient = "horizontal", command = canvas_image.xview) # defining horizontal scrollbar
    horizontal_scrollbar.pack(side = BOTTOM, fill = X) # placement of this scrollbar is at the bottom
    horizontal_scrollbar.config(command=canvas_image2.xview)
    vertical_scrollbar.config(command=canvas_image2.yview)
    canvas_image2.configure(yscrollcommand = vertical_scrollbar.set, xscrollcommand = horizontal_scrollbar.set) # this is where I assign the scrollbar command onto canvas_image specifically
    canvas_image2.configure(scrollregion=canvas_image2.bbox("all"))
    canvas_image2.config(highlightthickness=0)
    canvas_image2.pack(side=LEFT, expand=YES, fill=BOTH) # .pack() allocates a coordinate space for canvas_image
    generate_output = Button(new_root, text = "Generate output", command = writeDoc)
    generate_output.pack()
    new_root.mainloop()
new_window_button = Button(root, text = "Window to display output" , command = new_window)
new_window_button.pack()
root.mainloop()
```

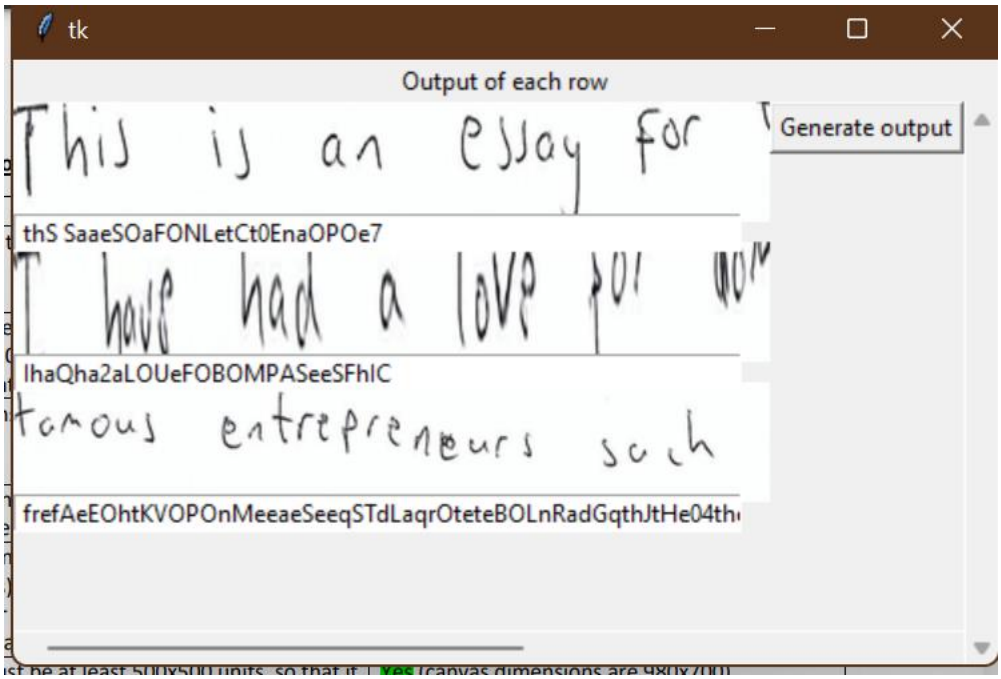
My project: Handwriting recognition research project

My User Interface:

Window 1 (user selection of input)



Window 2 (output of image):



Section 3.4.7) Review

Work I have done

In this section, I have mainly done 4 things:

1. Make GUI that allows the user to enter images for image recognition

2. Make a button that can detect letters in the user input image and output it to the user (using bounding boxes)
3. Make a button that outputs the results of my neural network recognition in a whole new window.
4. Make an output file for the output of the recognition program

I additionally made a Tkinter GUI for my testing purposes (for example, displaying num values for letters (unique identifiers that I assigned to letter images) on Tkinter window alongside bounding boxes, to monitor and observe the results of my partitioning algorithms (points 44 and 5))

Testing of my work

I have done all the necessary testing in Stage 3.

Has it met my key features of my solution (Section 1.5)/success criteria (Section 1.6)?

Key features (neural network)	Met/Not Met	Explanation/Justification
Dedicated widget to show input image	Met	I have made a widget that shows input image that the user has entered. I used Tkinter Canvas where I can place images and draw rectangles on it (for bounding boxes)
Accept only ".JPEG/JPG" and ".PNG" files	Met	I validated the user input, using Tkinter. filedialog.askopenfilename()) where I specified that only .png,/jpg, and .jpeg images are allowed.
Add zoom in and out operations for image	Met	I made a zoom slider using the Scale method in tkinter and thoroughly update the images and bounding boxes scale factors using Scale command function
Show bounding boxes around each detected letter	Met	I managed to feed in coordinate data from detect_letter.py using which I created Canvas create_rectangle() which became my bounding boxes
Show snippet of each row of input with text output beneath it.	Met	I have used my knowledge in list slicing to crop out images of each row of input and display it in a new window (using Tkinter Toplayer() method) with input boxes below it for the user to add any amendments

Make an output text file	Met	I used Entry.get() method to fetch string values for each row and save it in a text document

Steps to improve it

When someone makes a Graphical User Interface, there is a lot things they must ensure are done correctly, such as user validation, finding bugs, and fixing bugs that can break my program. I feel like I should have spent more time finding and fixing bugs in my GUI code because there can be scenarios that the client keeps on getting errors, which is a bad user experience.

Section 4) Evaluation

Section 4.1) Post Development Testing

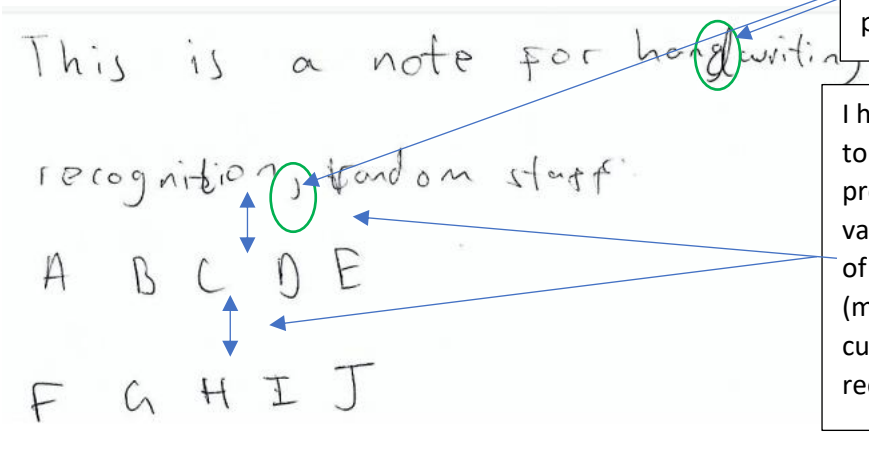
First of all I must test the Post development data that I wrote about in **section 2.8**

Test	Input data	Desired outcome	Actual outcome	Pass/Fail	Explanation/Justification
I will input a .JPEG/JPG/PNG file containing handwritten data on a plain white background to test how accurate the output of the GUI is for the given data	**Image 1 in input images (with annotations)**	Since this is the post development testing, I must critic my work, so I expect most of my letters in the image to be recognised correctly, and the output file to accurately map out the image into text.	Image 1 in Actual output for test table above	Pass (barely)	I think my program passed my test, although I am disappointed because not all the letters in this image (which had no noise in it and had a white plain background) were detected. On the other hand, my program recognised that there were 4 lines in this image and gave 4 outputs (1 for each row).
I will input a .JPEG/JPG/PNG file containing handwritten data on a lined paper background to	**Image 2 in input images (with annotations)**	I expect that my program eliminates excess noise from image and outputs a	Image 2 in Actual output for test table above	Fail	I think that the overall letter outputs of my program were highly erroneous for this image, hence why this has failed my test

test how accurate the output of the GUI is for a noisy image such as lined paper		text file that accurately maps out the input into text.			
I will input a .JPEG/JPG/PNG file containing cursive handwritten data to assess how the program will handle erroneous data and convert it to digital text	Image 3 in input images (with annotations)	I know from past experiences that cursive data will not be detected by my program at all, so I do not have any high hopes for my program to give a meaningful output	Image 3 in Actual output for test table above	Fail	I expected this test to fail. This is because I know that my letter detection algorithm cannot tell apart letters inside of cursive words (because there is no spacing between them).

Input images (with annotations):

Image 1 :

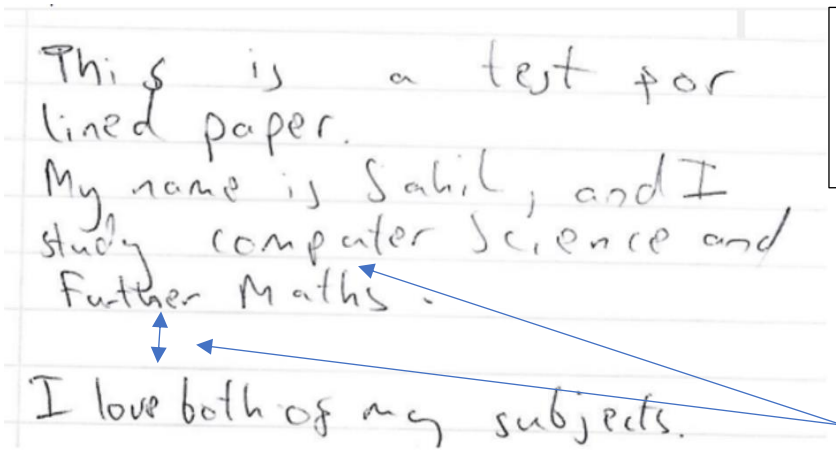


I made deliberate false errors such as making a bad looking "d" and using punctuations (which my program cannot recognise) so I can observe how my program tackles these problematic scenarios

I have tried to make this image difficult to read to check for robustness of the program. For example, I have put varying sizes of gaps between each line of writing to introduce irregularity (modelling real life scenarios). I am curious whether my program recognises the 4 rows

Image 2:

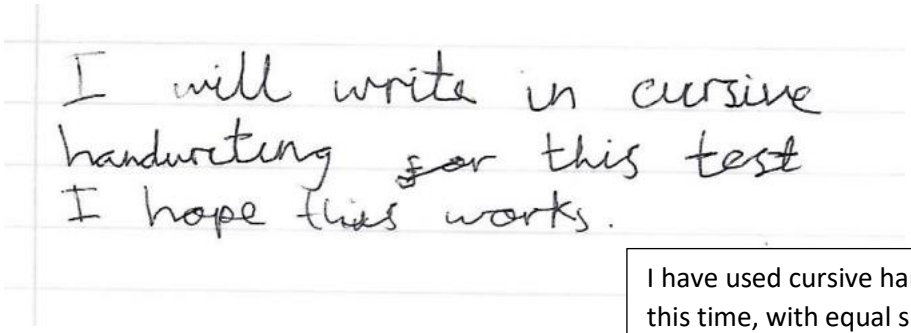
My project: Handwriting recognition research project



I have used lined paper, so I can monitor what influence noise can have on my recognition program.

Again, I have tried and put varied spacing between each row to monitor how my program will process this situation (non-linearity)

Image 3:



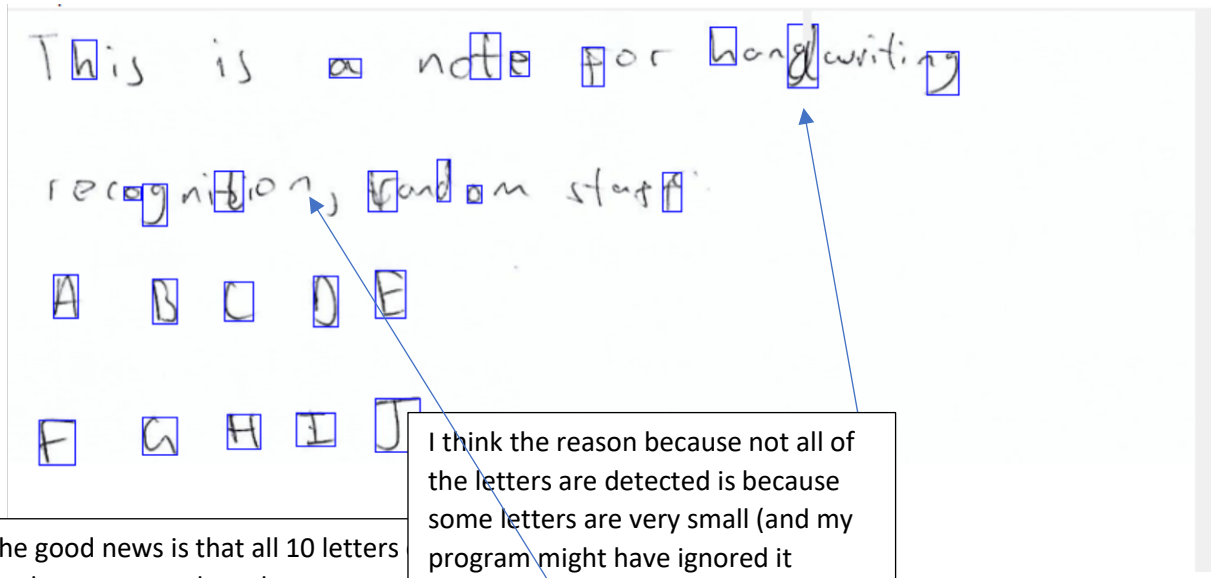
This time I have been much more linear in my row spacing because checking the spacing is not the purpose of this specifie test

I have used cursive handwriting this time, with equal spacing between each word

Actual output for test table above (with annotations):

Output for Image 1:

This is the window where my program detects letters



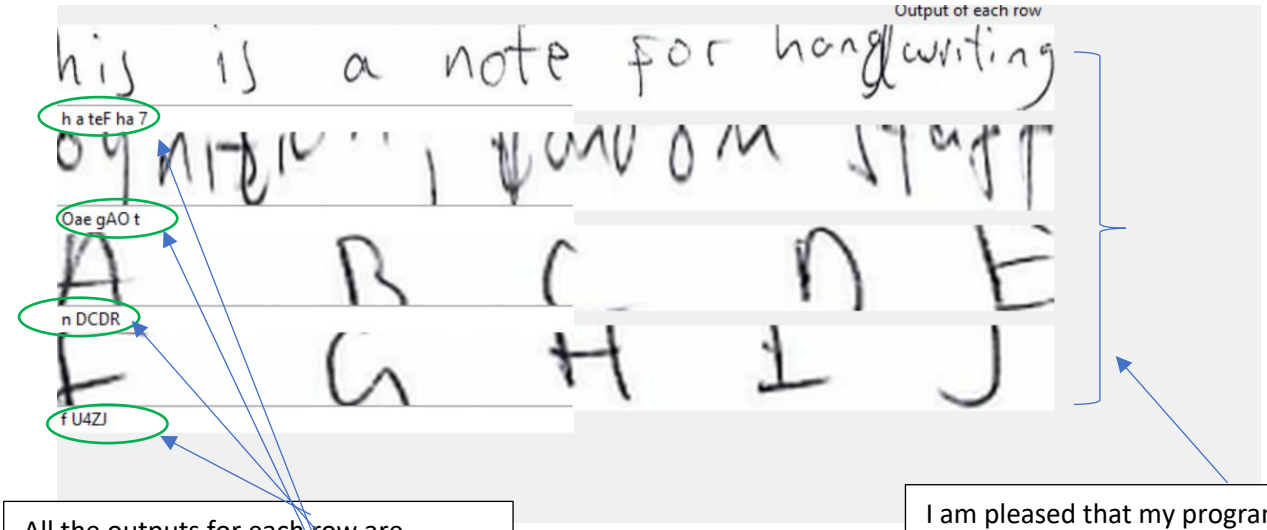
The good news is that all 10 letters the last two rows have been detected.

I think the reason because not all of the letters are detected is because some letters are very small (and my program might have ignored it thinking it is noise). Also, since my detection algorithm uses binary thresholding, some faint letters might have been erased as well because it didn't meet a threshold value

ogram hasn't
ed all of my letters
disappointing.

My project: Handwriting recognition research project

Output for the Recognition window:



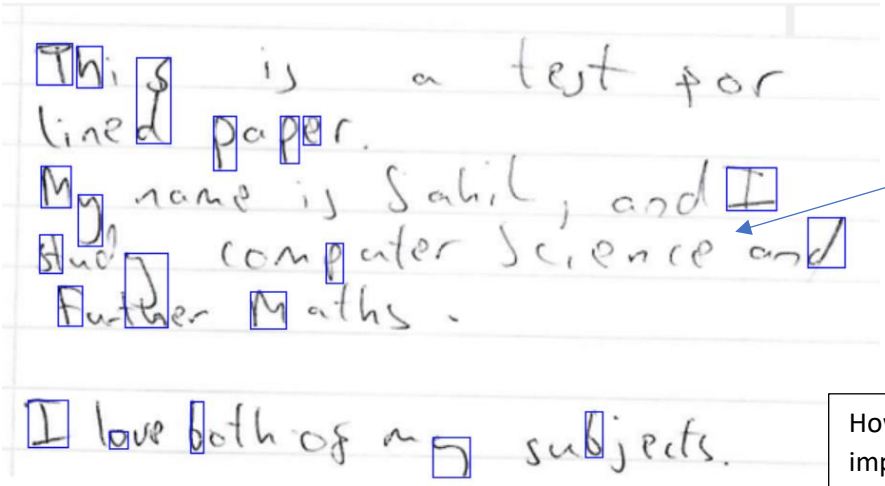
All the outputs for each row are random which is disappointing, meaning the user experience of the GUI for recognising images will be low.

I am pleased that my program successfully recognised the fact that there are 4 rows in this image and the cropping of each row is accurate.

Output for Image 2:

This is the window where my program detects letters:

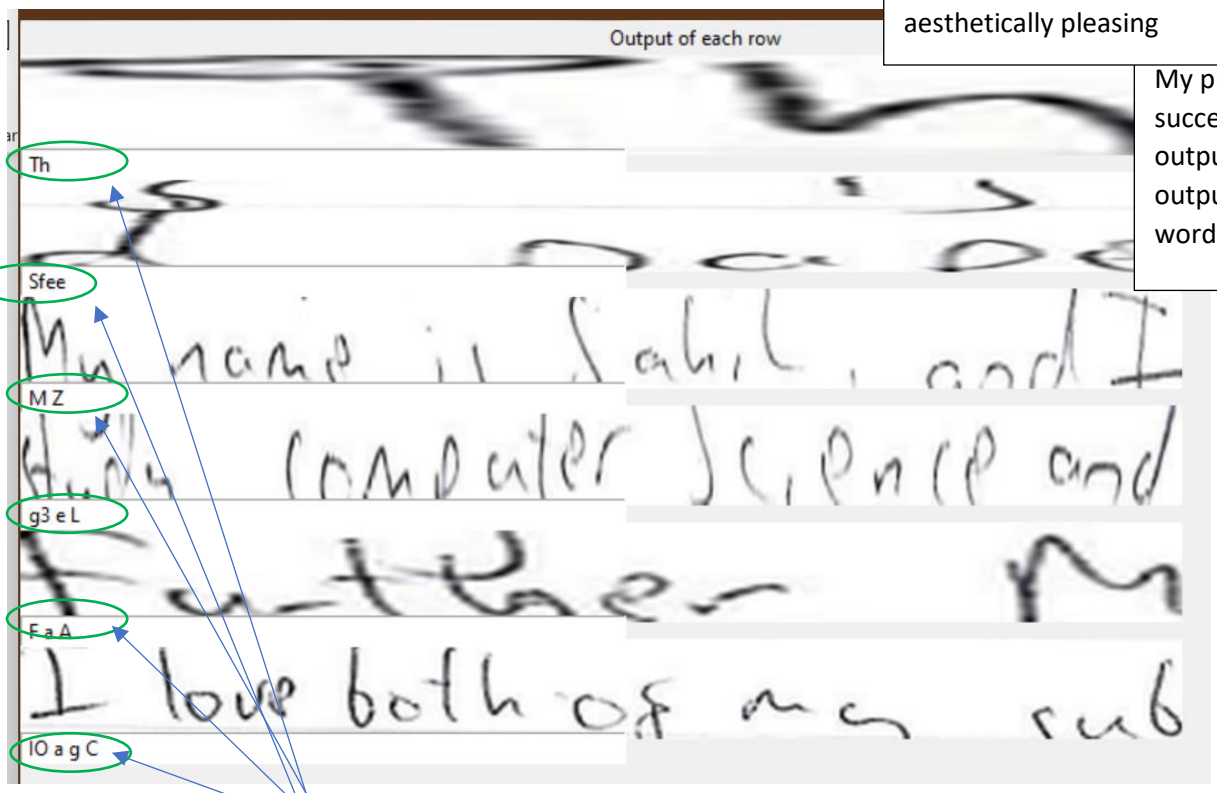
My project: Handwriting recognition research project



As with image 1, my program hasn't detected all the letters in image 2 as well.

However, I am impressed because the program managed to eliminate noise (i.e. not detect lines as letters) so no false letter images are passed to the neural network

Output for the Recognition window:



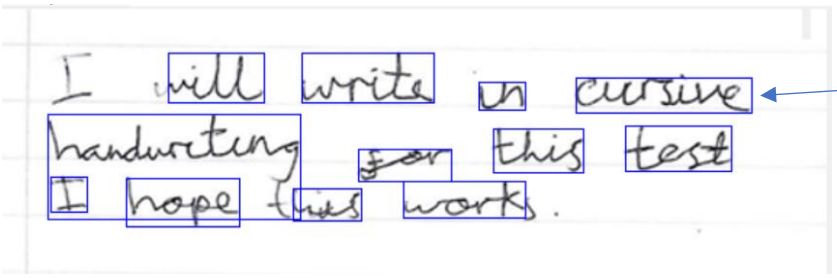
The scaling of my rows is very irregular, and it does not look aesthetically pleasing

My program successfully outputted 6 output rows for words.

The outputs of all the rows is very underwhelming to me because I am getting 1 word results where I should be getting a full sentence of words. Also, the words being output are very random and erroneous

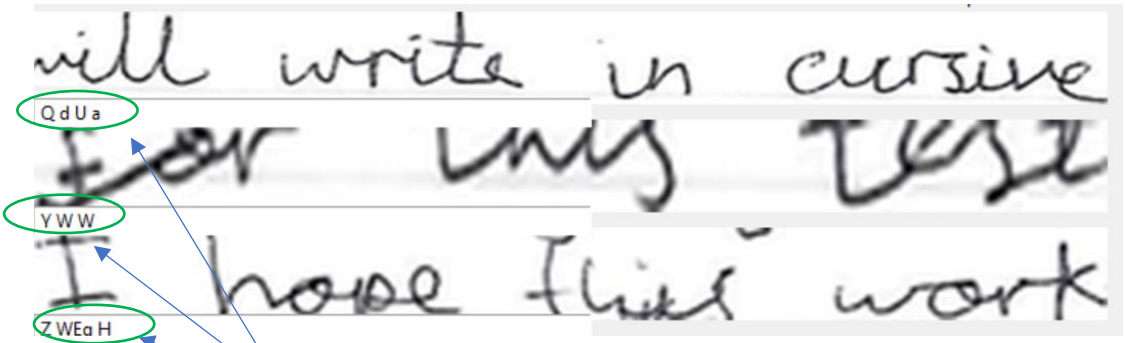
Output for Image 3:

This is the window where my program detects letters:



My detection algorithm has detected words instead of each individual letters inside words

Output for the Recognition window:



I am pleased that my program detected there are 3 rows in this image

The results for each row are completely erroneous.

Section 4.2) Usability testing

In my Tkinter GUI, I think I have added all my usability features from Section 2.4,

Testing table

Test	Input data	Desired output	Actual output	Pass/Fail	Explanation/Justification
Large dimensions for image widgets	Run the program and observe the size of image widgets in first and second window	Large widgets where the image can be fully seen	Large widgets where the image can be fully seen	Pass	I designed my user interface specifically around large image widgets because I know that the users would want to the viewing experience of the image to be good and that isn't possible if the widgets are small, so I needed to implement big widgets
Scrollbars for all image widgets	Run the program and use the horizonta	Horizontal and vertical scrollbars both work	Horizontal and vertical scrollbars both work	Pass	I know that scrollbars will be necessary in my program because not all images are small, some images are very big and

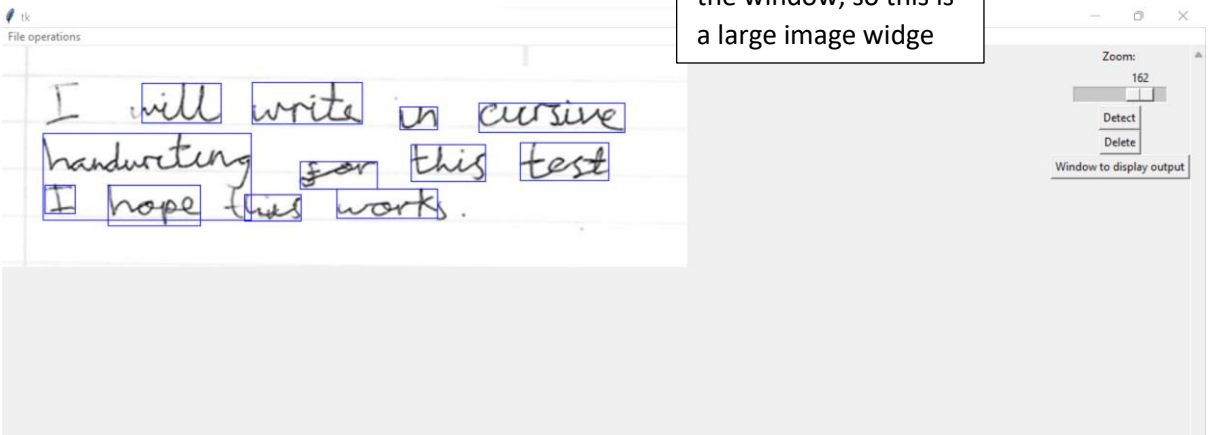
	l and vertical scrollbars in both windows	properly (i.e., allow navigation of large images)	properly (i.e., allow navigation of large images)		to navigate big images, I will need scrollbars
Consistent and plain colour scheme	Run the program and observe the colours of the user interface	Plain colours used in the user interface and colour scheme is consistent with all windows	Plain colours used in the user interface and colour scheme is consistent with all windows	Pass	Plain and consistent colour scheme give my program a "professional and minimalist" look and feel, because my program isn't trying to exaggerate by using any bright colours but instead just using the colour white which is plain and not too out of the ordinary
Large button dimensions	Run the program and observe the sizes of buttons	Large buttons for both windows	Small rectangular buttons	Fail	Unfortunately, this is a usability feature that I forgot to implement. I should have made my button dimensions bigger.

Input data for table above:

Test 1:

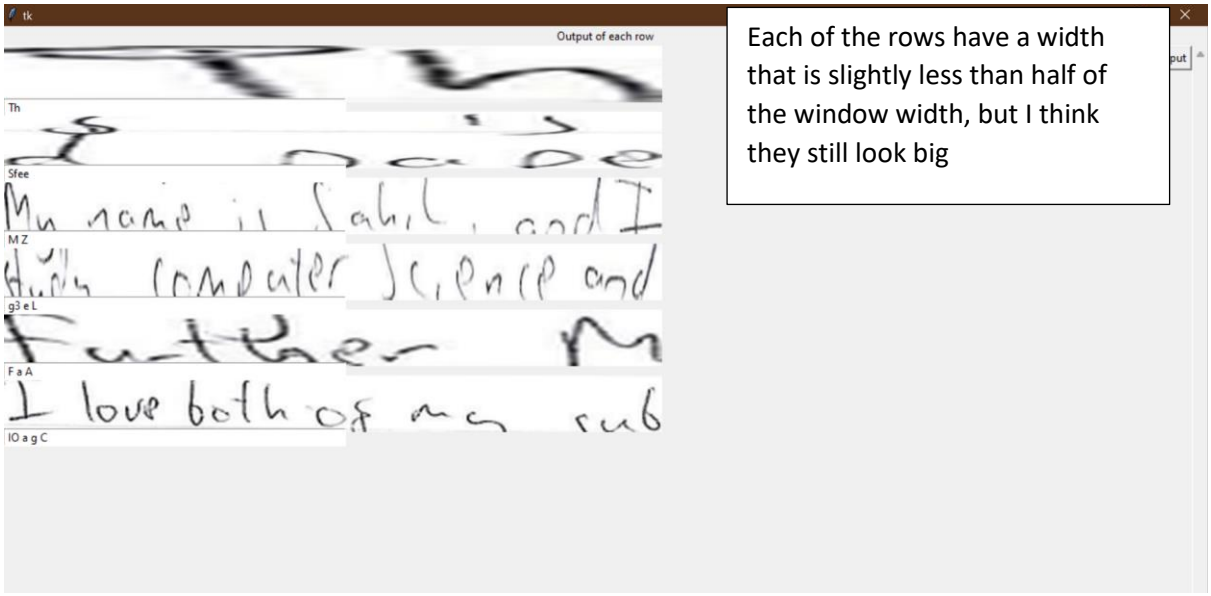
User input window

The width of the image widget is more than half the width of the window, so this is a large image widge



Recognition window:

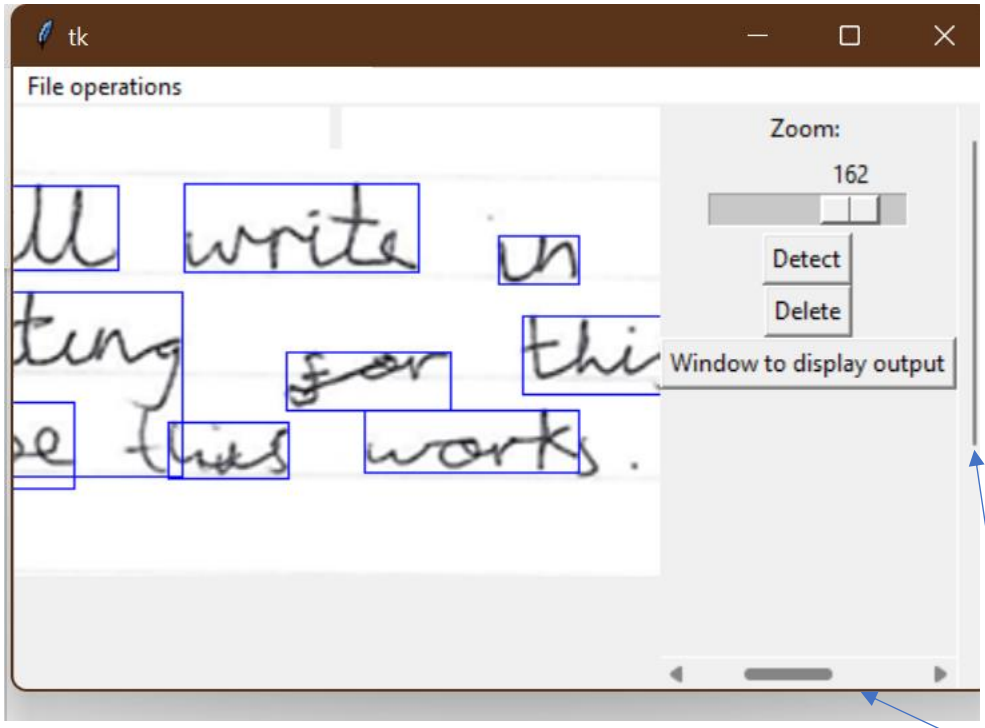
My project: Handwriting recognition research project



Each of the rows have a width that is slightly less than half of the window width, but I think they still look big

Test 2:

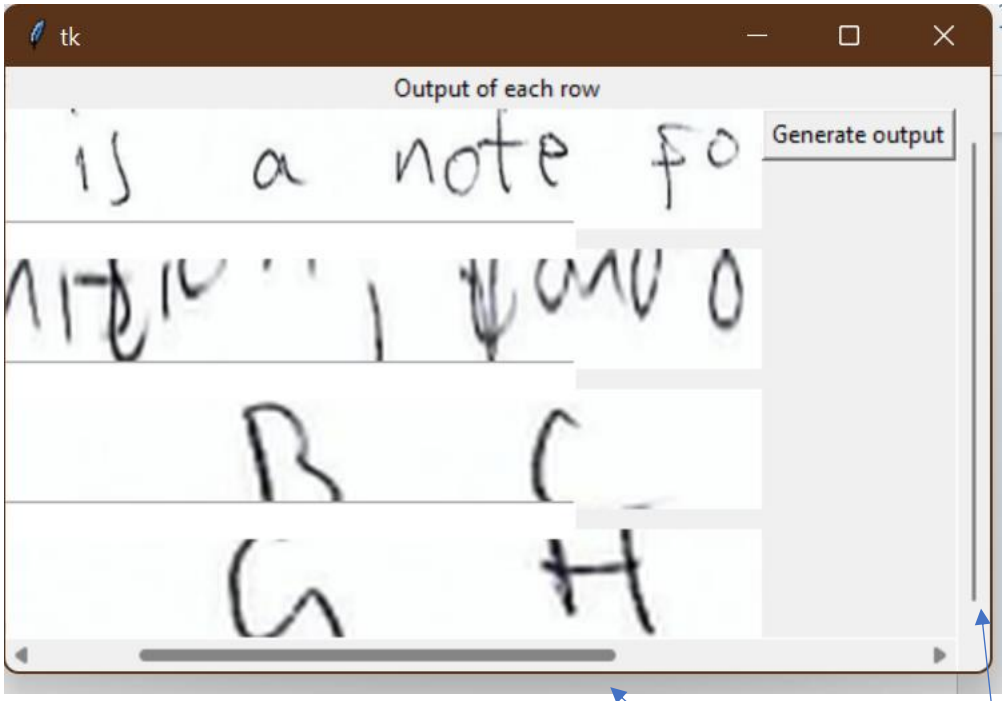
User input window



The horizontal and vertical scrollbars both allow to navigate any input image

Recognition window:

My project: Handwriting recognition research project

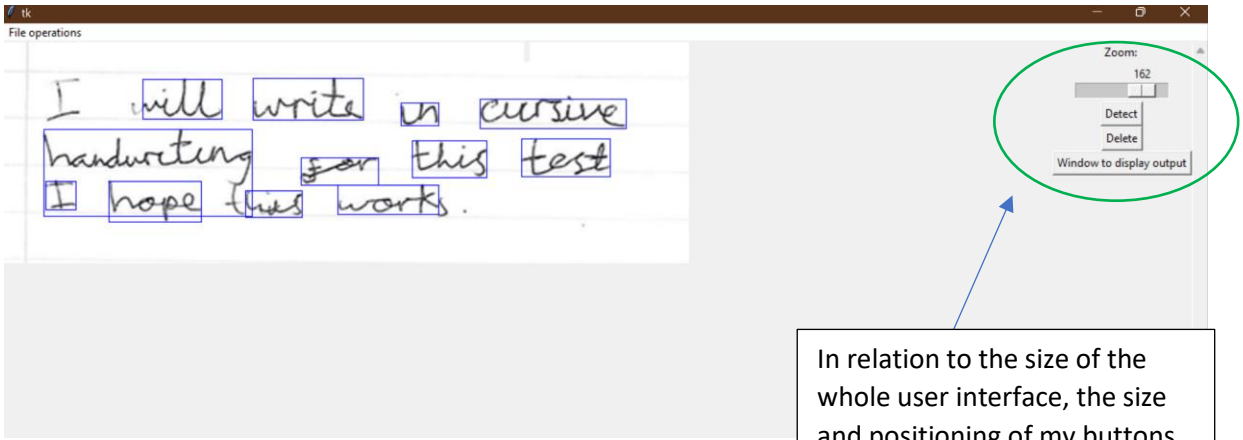


The horizontal and vertical scrollbars both allow to navigate any input image

Test 3:

From the above images, you can deduce that all the colours used in my GUI are consistent.

Test 4:



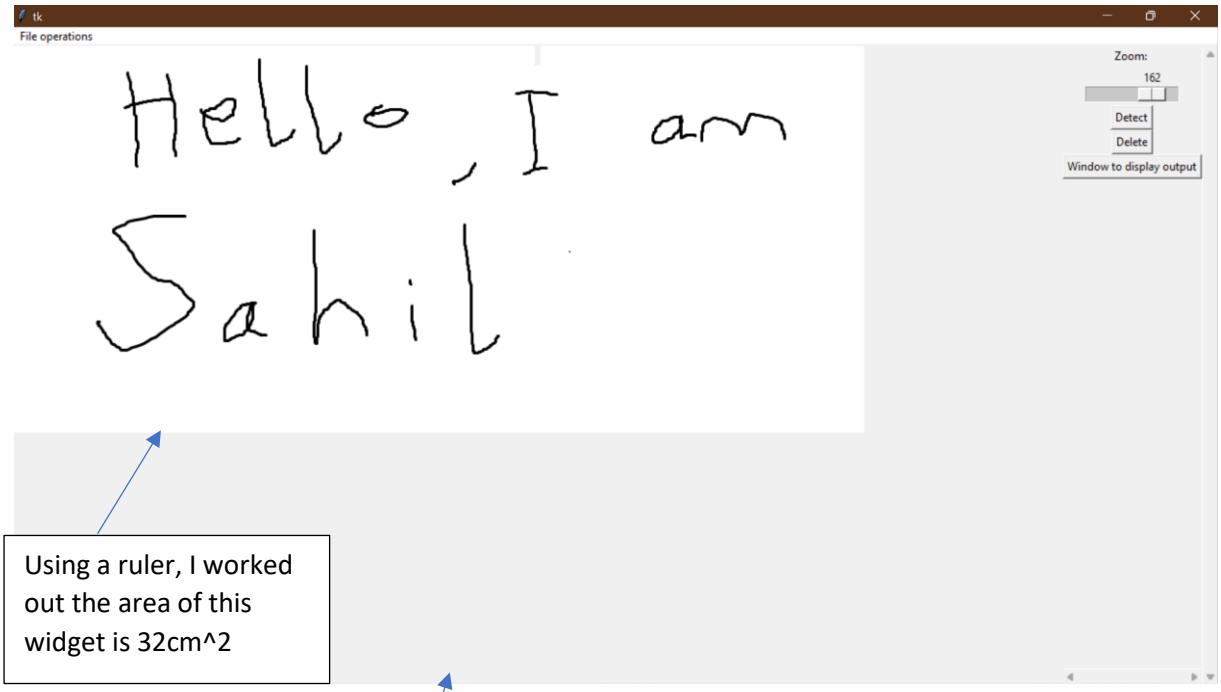
In relation to the size of the whole user interface, the size and positioning of my buttons is not right. The sizes of the buttons can be bigger, and there should be more spacing between each button

Section 4.3) Evaluation for Usability features

Objective	Met	Evidence
Large dimensions for image widgets	Met	Image on page 209
Scrollbars for all image widgets	Met	Image on page 209
Consistent and plain colour scheme	Met	Objective 9 on page 210
Large button dimensions	Not Met	No evidence

Evidence of usability features with justification:

Objective 1:



Using a ruler, I worked out the area of this widget is 32cm^2

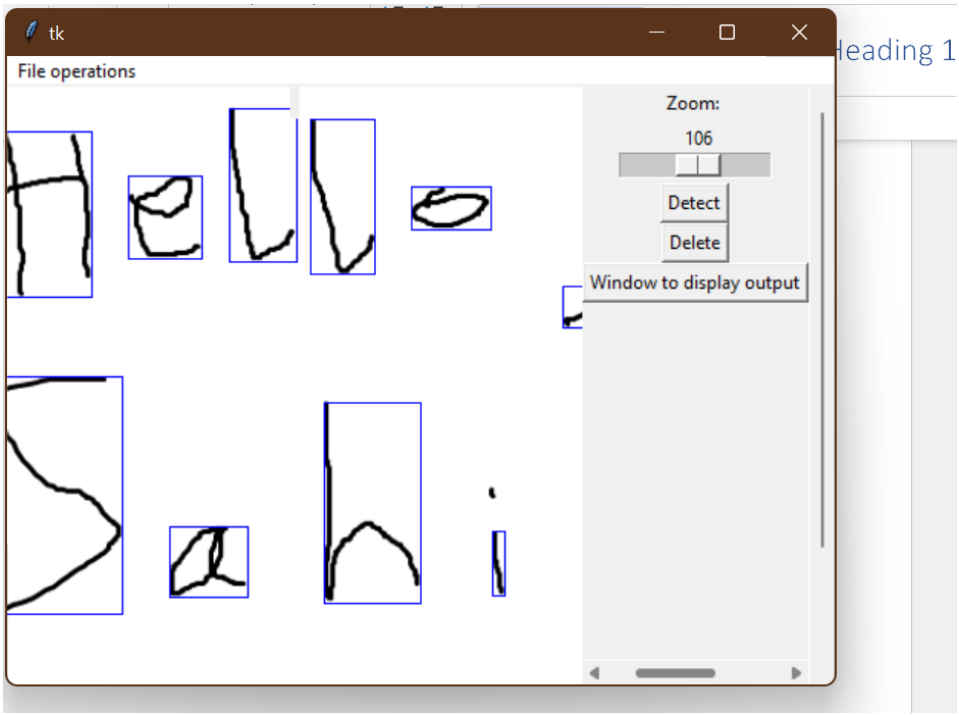
Using a ruler, I worked out that the area of this window is 66 cm^2

Using maths, I worked out that the image widget takes up roughly 50% space of the whole window, which is great for user experience because there will be no compromises for the user to view their images (e.g., if the viewing widget is too small then not a lot of the image can be seen).

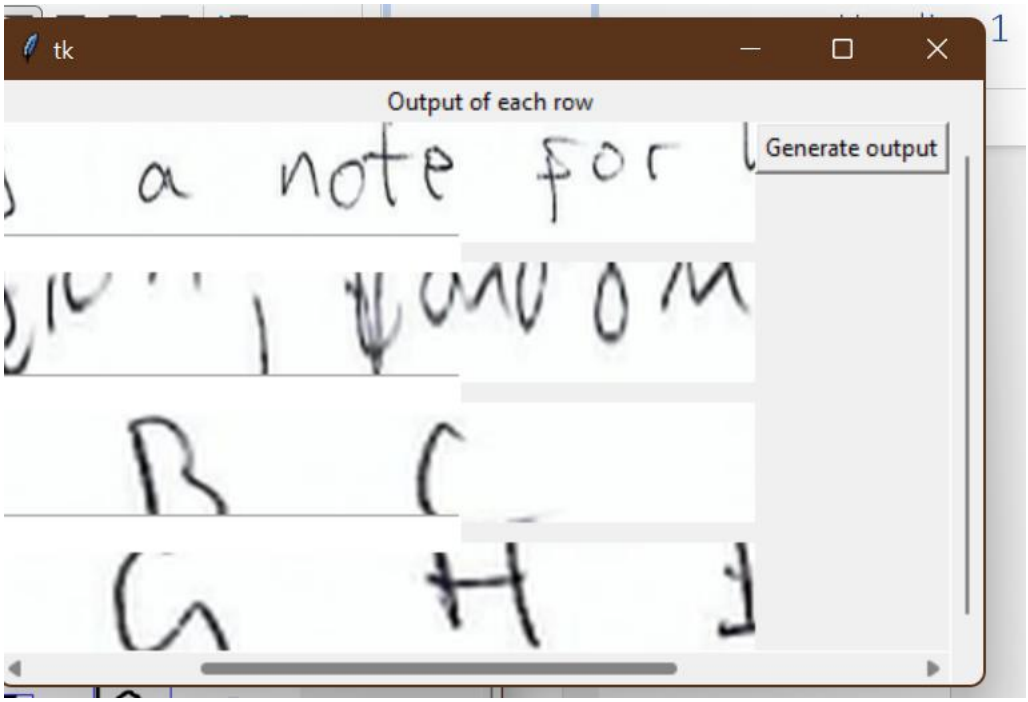
Objective 2:

My project: Handwriting recognition research project

First window:



Second window:



Both of my windows have scrollbars to navigate image regions. This is useful in terms of usability features because it can help navigate the user through large images where getting to certain regions is hard.

Objective 3:

The only color that my program has is gray (the default window colour). I haven't experimented with colors in my program because I wanted to design a simple and minimalist user interface.

Objective 4:

I hadn't implemented larger button sizes in my program, nor have I paid attention to the positioning of each button. This is bad for usability features because smaller buttons can be less easy to use, because they are small and be hard to locate. In the future, I will spend more time on building my user interface (in terms of aesthetics) so it can contribute to good user experience.

Section 4.4) Evaluation of Success Criteria:

Objective	Met?	Evidence
Having a clean user interface by having no more than 4 buttons on one window	Met	Test 2 on page 201 (clean interface with only 3 buttons)
Have 2 tabbed windows	Not Met	No evidence
Load a dropdown menu where the user can choose to open an image	Met	Objective 3 on page 205
Have a widget that covers 50% or more of the window that shows the input user image	Met	Objective 1 on page 204
Have a zoom functionality to enlarge input image to 200% or shrink to 0 % (inside widget)	Met	Test 2 on page 201
Have scrollbars (horizontal and vertical) to navigate user input image in widget	Met	Test 2 on page 201
Have a 1 button to detect letters in the user input image	Met	Test 4 on page 202
Have 1 button to output digital text generated by the CNN (redirects to a new tab)	Partially Met	(My button doesn't redirect to a new tab, but a new window) Objective 1 on page 203
In the output, display snippets of each row of input text (if there were paragraphs in input text) alongside generated text output just below. It should cover at least 40% of the screen	Met	Test 2 (Recognition window) on page 202
Have scrollbars (horizontal and vertical) to navigate rows in output window	Met	Test 2 (Recognition window) on page 202

My project: Handwriting recognition research project

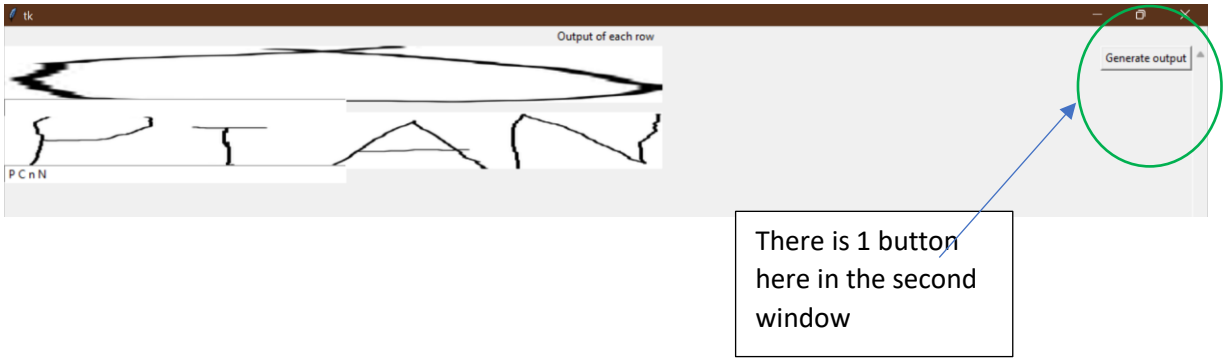
Generate a .txt output file on a button click	Met	Top of page 212

Evidence and justification for objectives:

Objective 1:

First window:

Second window:



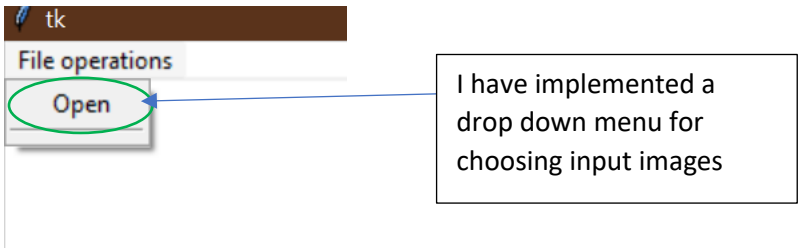
Having no more than 4 buttons in a window makes the user interface look lightweight and clean, as opposed to having a lot of buttons and a lot of function implementations that can make the interface look overwhelming and complicated because there would be too many functions the user would have to learn. I think my interface is very easy to use and welcoming to any user because it doesn't have a saturation of buttons, or widgets.

Objective 2:

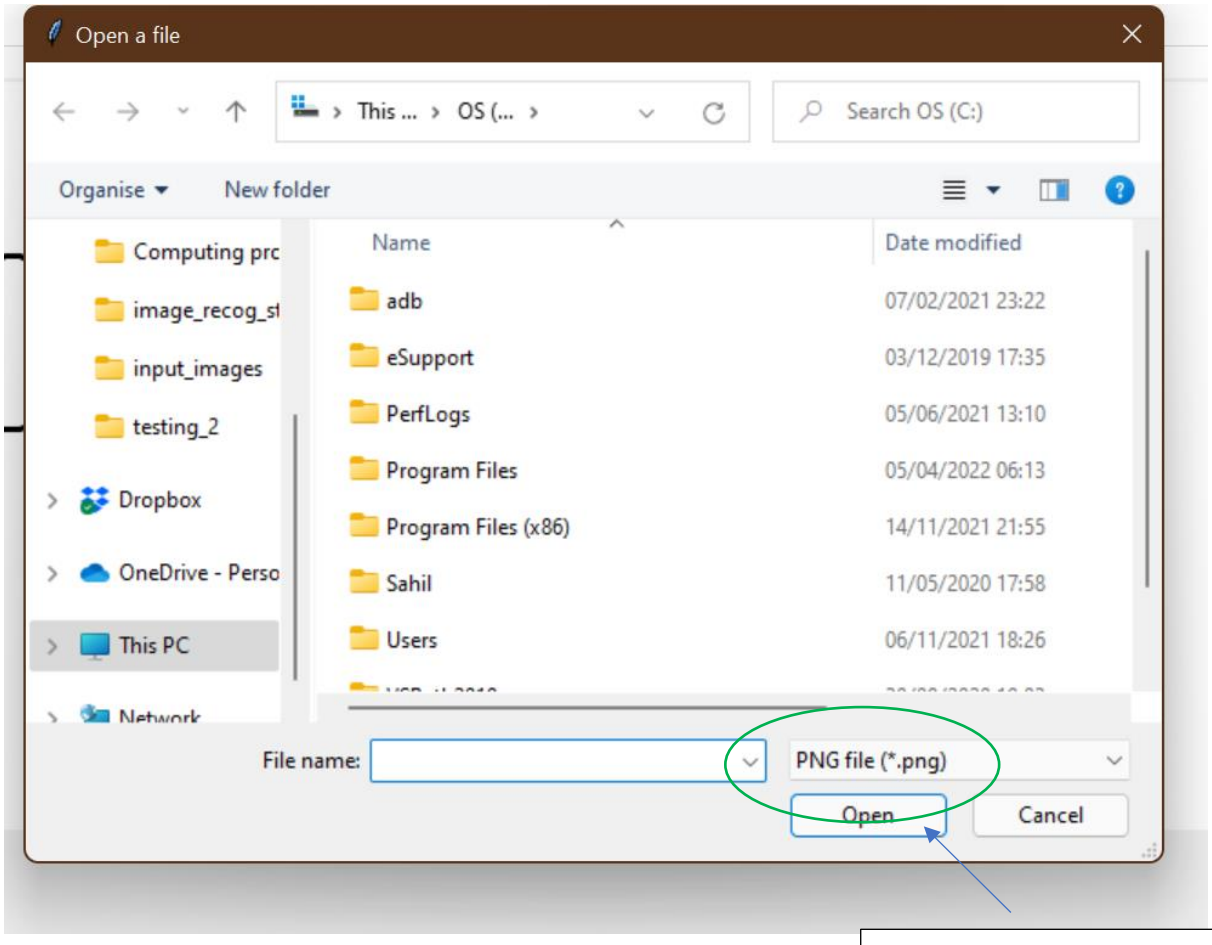
I have not met this success criteria. This is because I personally found it really hard to implement because when I looked for any solutions online (for making tabbed windows) they all formatted their code in a very different way to mine. This meant that I would have

to rewrite and change my code to facilitate tabbed windows, which is time consuming. In the future, I will do detailed research on all my desired functions for a solution first, and then start writing code so my code can fit in with the correct approach of implementing a desired feature. Not implementing tabbed windows has also negatively impacted the usability of my program because tabbed windows would make my program look more organised and clean, but using traditional windows can cause inconvenience while switch tabs, (as you would have to navigate down to taskbar and manually change it, which can be inconvenient is some cases).

Objective 3:



When the button is pressed, the user is taken to this window:



Using a drop down menu and a window where the user can sel
friendly, as opposed to typing out the path of the image, becau
quickly navigate their directories and select their image without
errors", whereas if the user types out a path, they might mistyp

in the right format (e.g., using relative path instead of an absolute path). Therefore, my approach for user input is justified.

Objective 4:

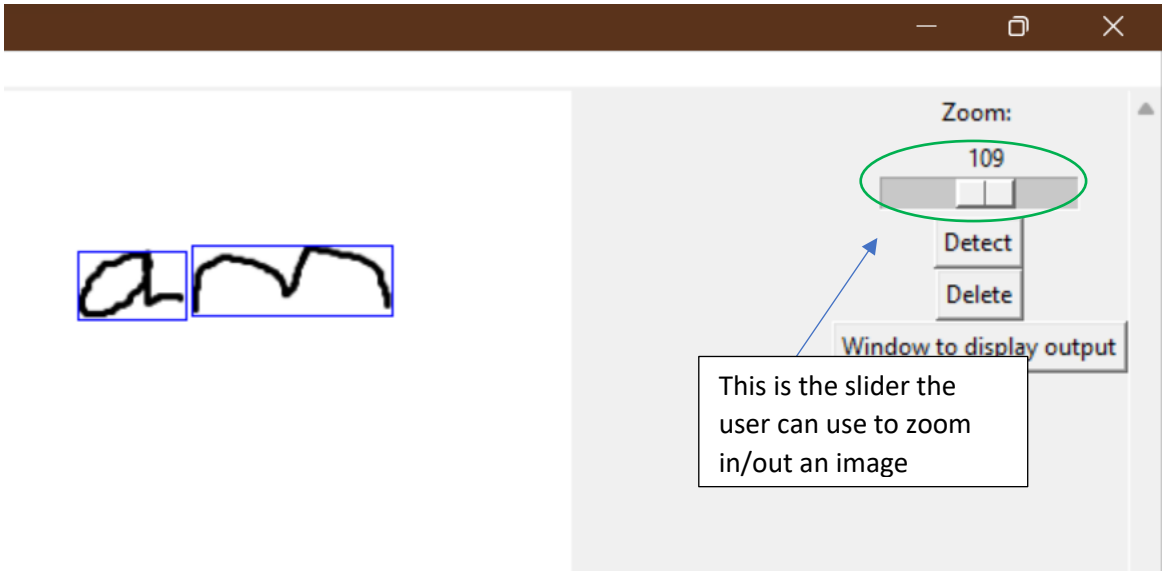
Using a ruler, I worked out the area of this widget is 32cm^2

Using a ruler, I worked out that the area of this window is 66 cm^2

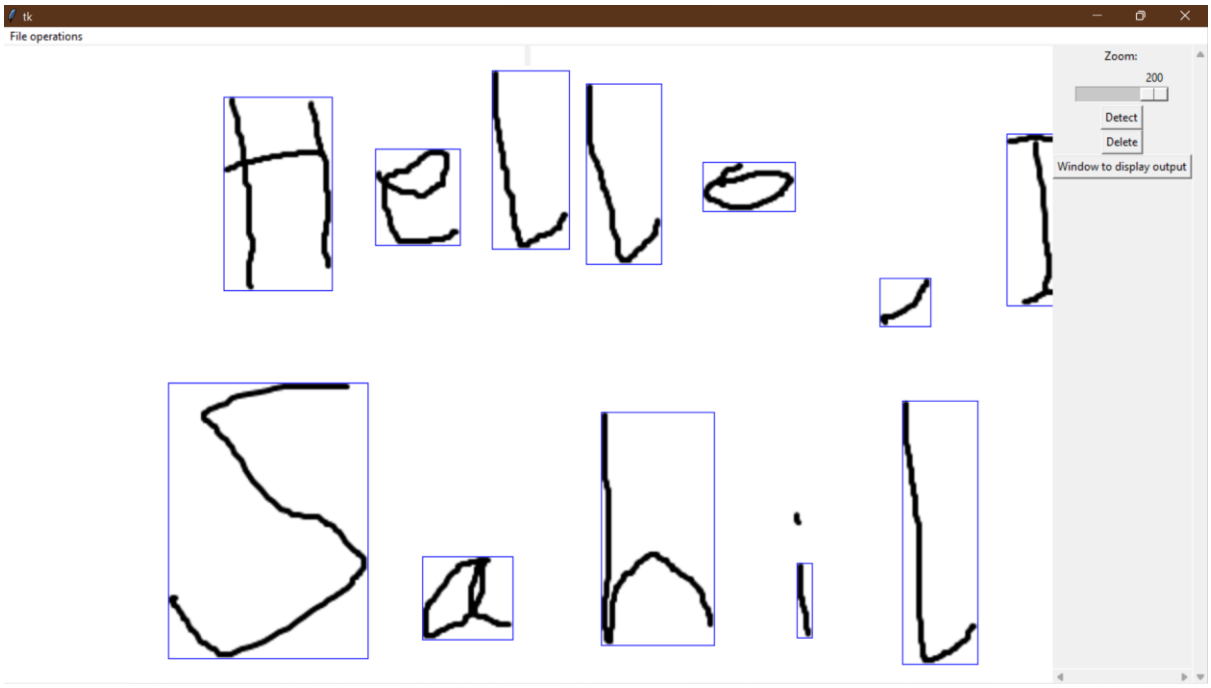
Using basic maths, I worked out that my image widget covers an area of 48% in relation to the image, which is roughly 50%, so my success criteria objective holds true. Having a large image widget would mean that the user can have a good viewing experience when using my letter detection function, etc. which creates good user experience.

Objective 5:

My project: Handwriting recognition research project

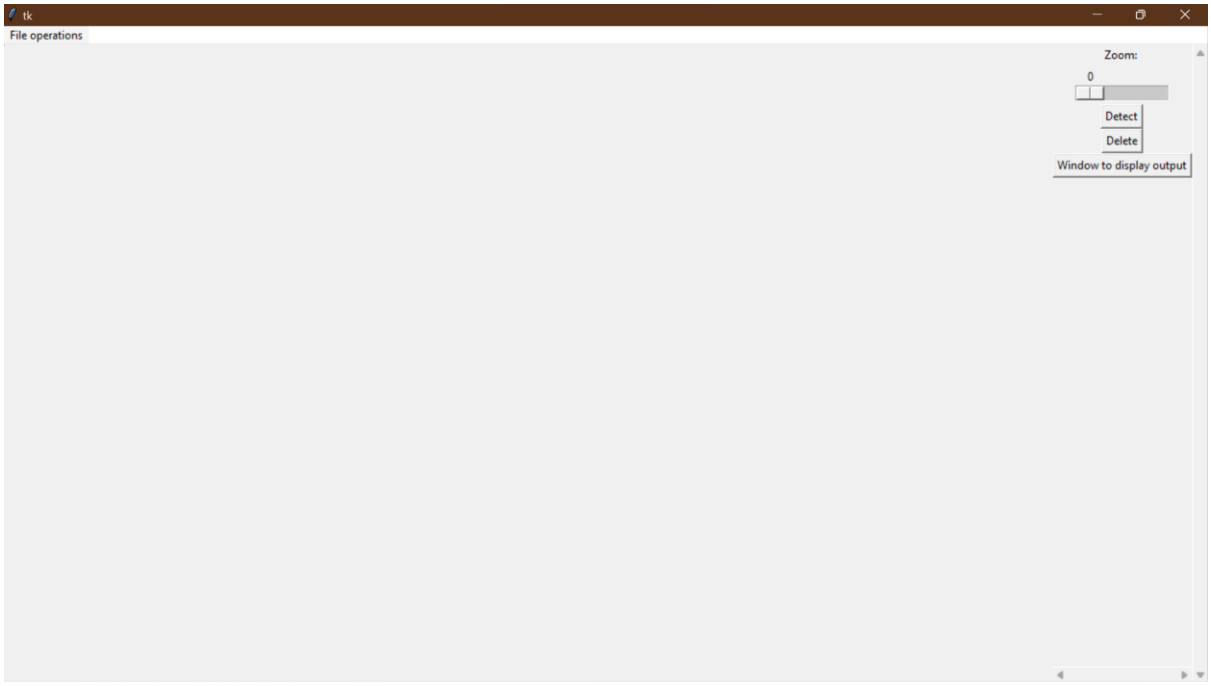


Zooming to 200%:



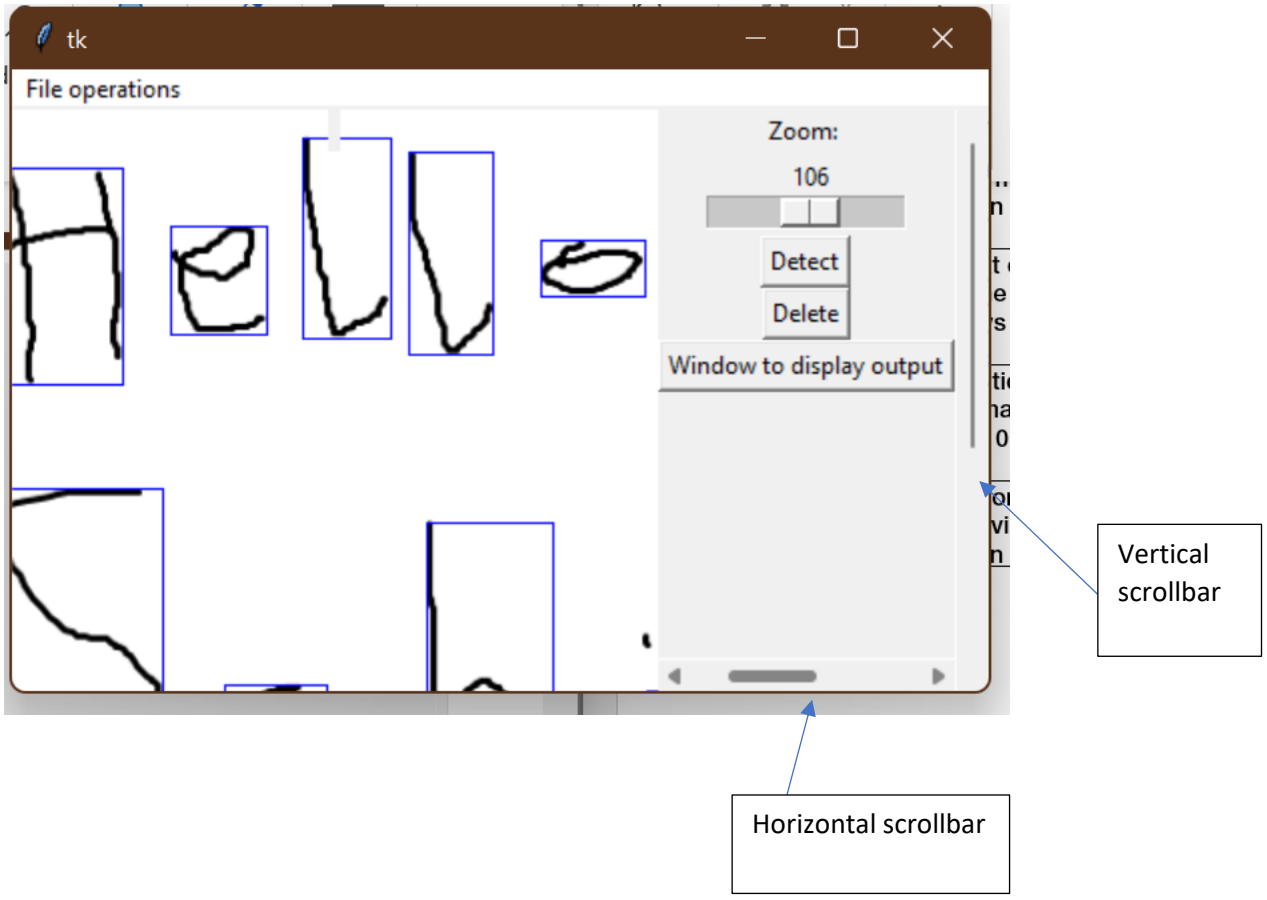
Zooming to 0%:

My project: Handwriting recognition research project



The zoom function is really integral to the experience of the user while navigating the interface. It can be used to make a small image (its original dimensions) larger and a large image shrink to accommodate the viewing experience.

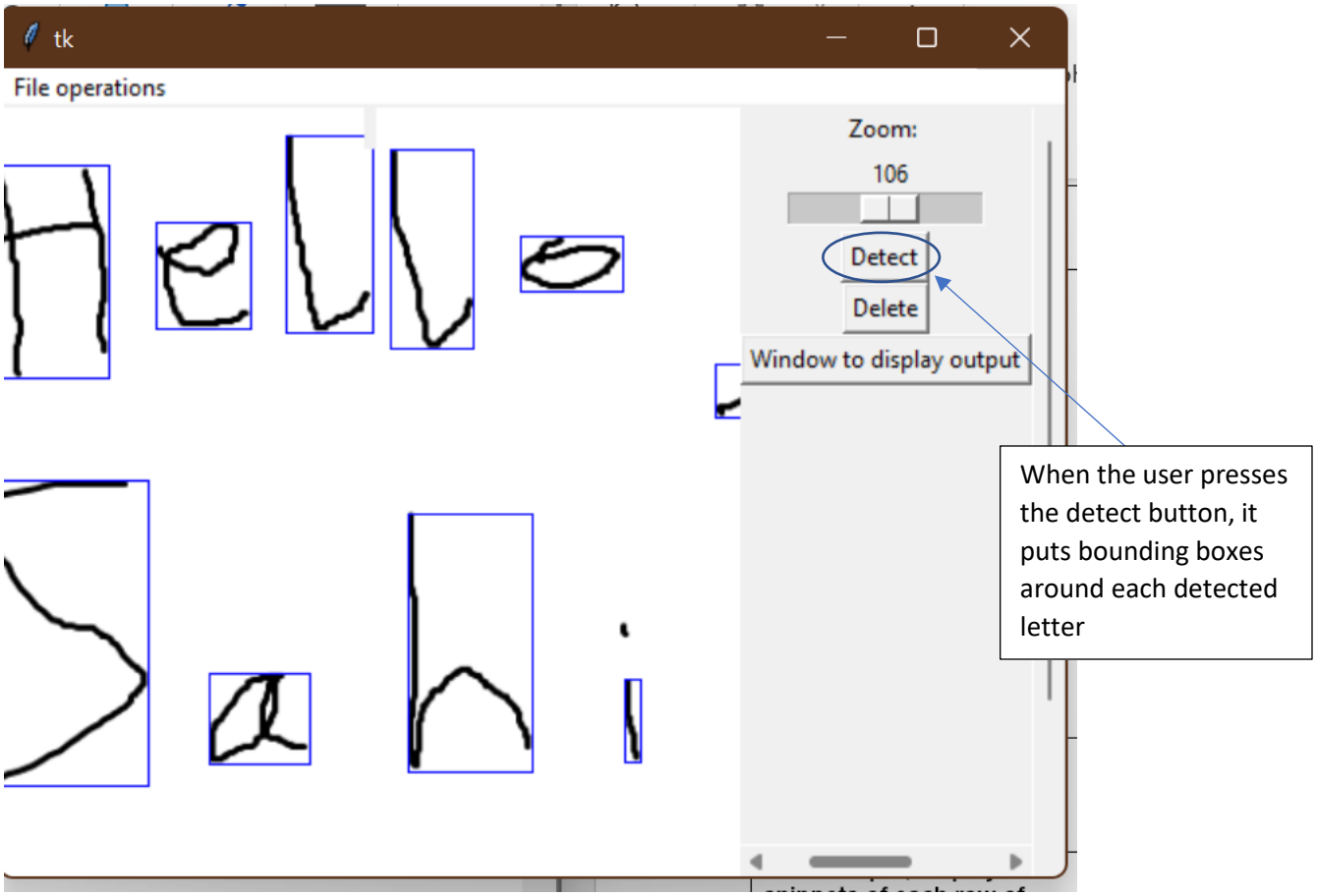
Objective 6:



My project: Handwriting recognition research project

Using a scrollbar to navigate large images is very useful, because it allows the user to view all parts of their image, or it can be used when zooming in to images to navigate remote regions.

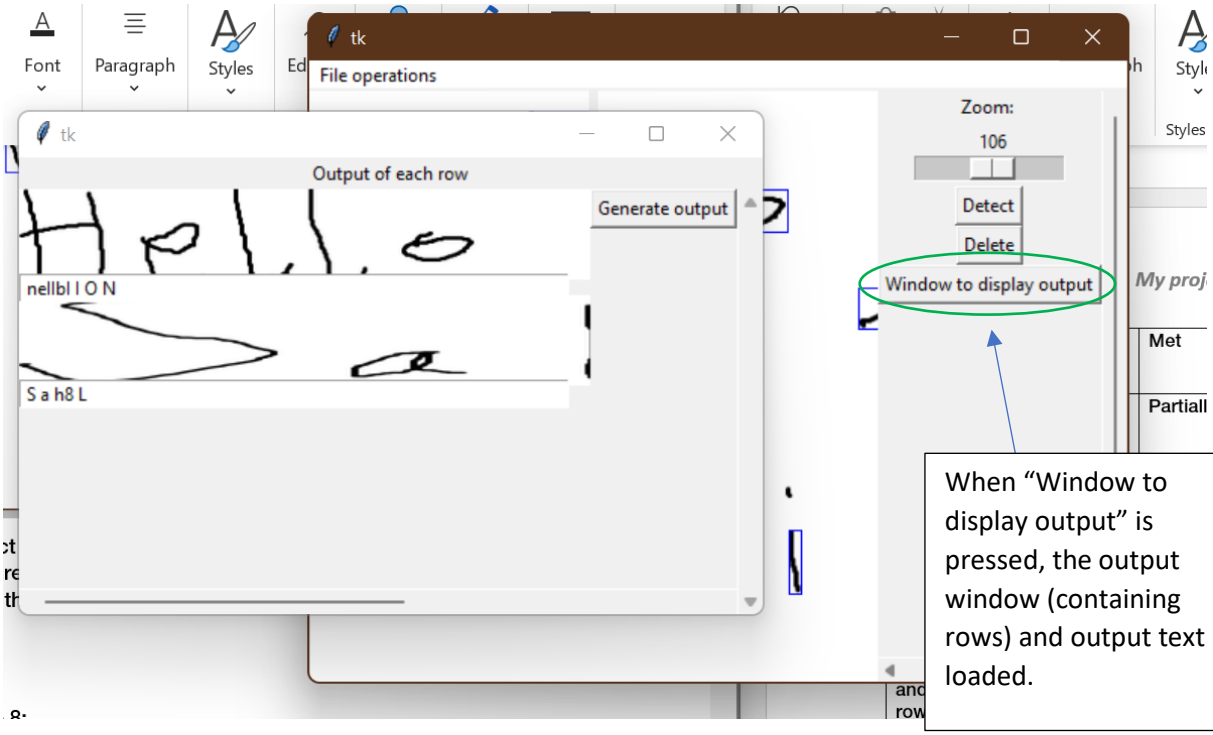
Objective 7:



The detect button is very useful because it shows the user what parts of their text regions have been detected, if some parts of the image aren't detected then they can rescan/rewrite image to make it detectable.

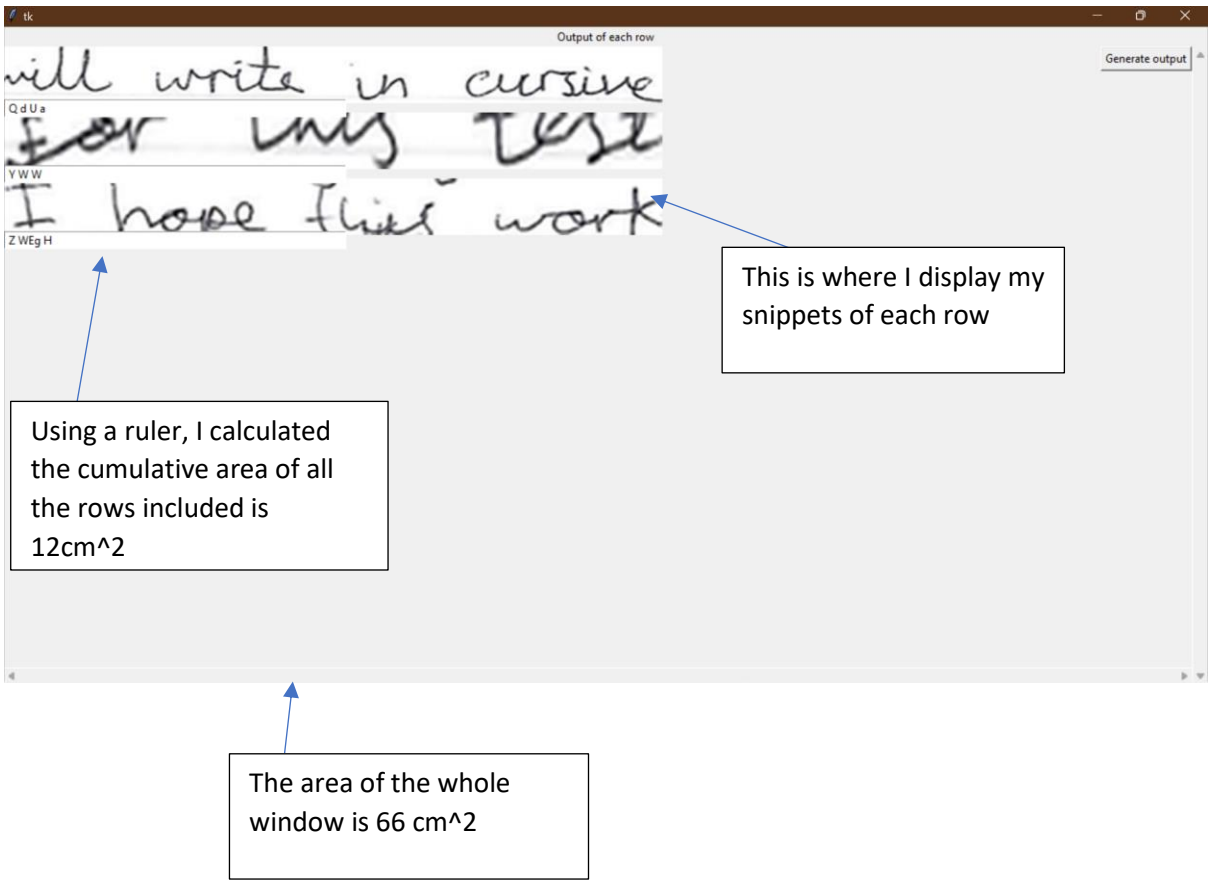
Objective 8:

My project: Handwriting recognition research project



I have partially met this objective because in the success criteria I have specifically said "redirect to a new tab", but since I haven't implemented tabbed windows, I cannot meet this objective.

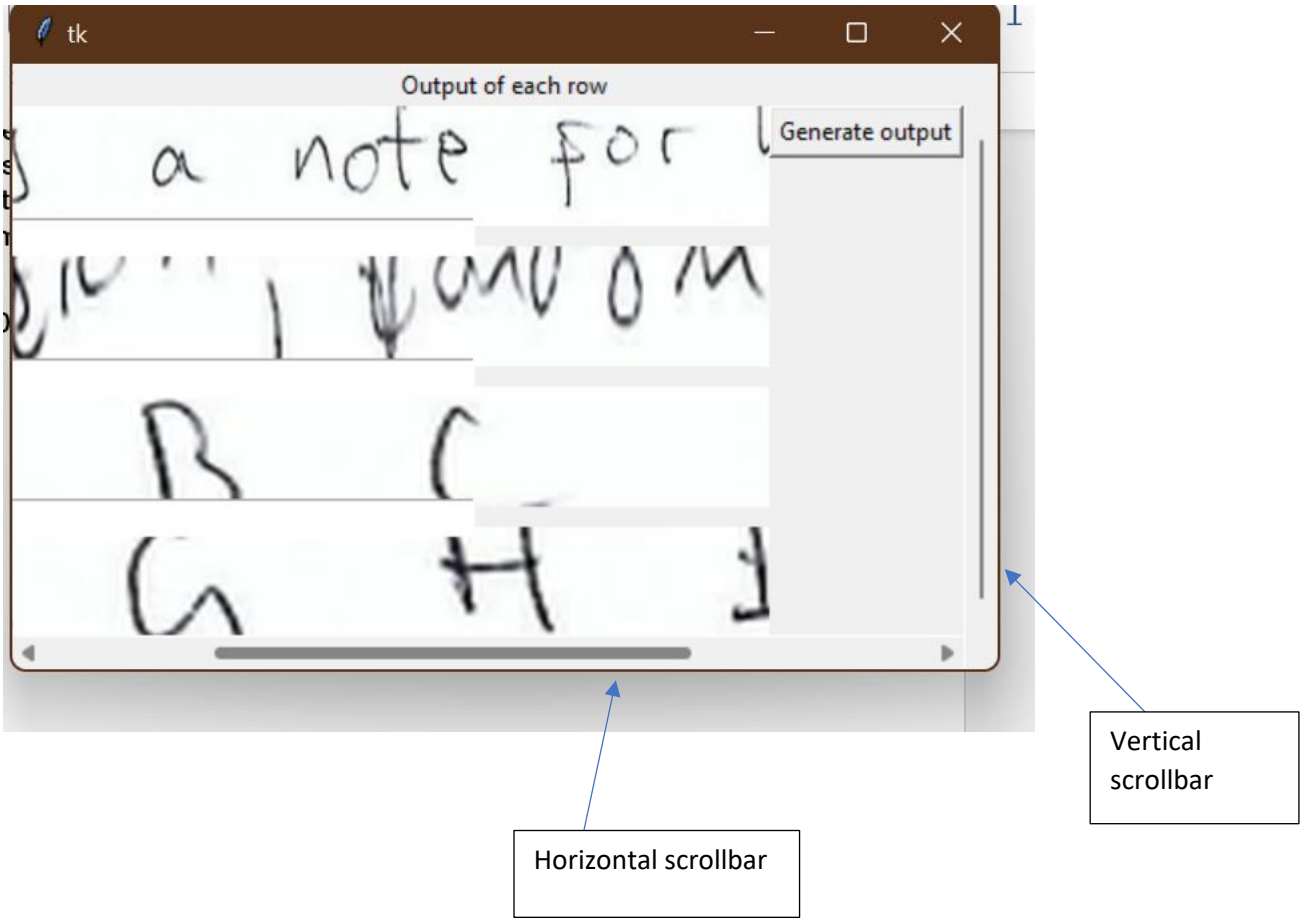
Objective 9:



My project: Handwriting recognition research project

The percentage of total window area that the images of rows take is 18% which is less than the 40% window area I said in my success criteria. However, I cannot stretch the row images too much otherwise it severely damages their image quality and it won't be useful to view. I think using row snippets is very useful for the user because it allows them to make changes to the output on the go, meaning they don't have to search for their input image again because the image information is in front of them already.

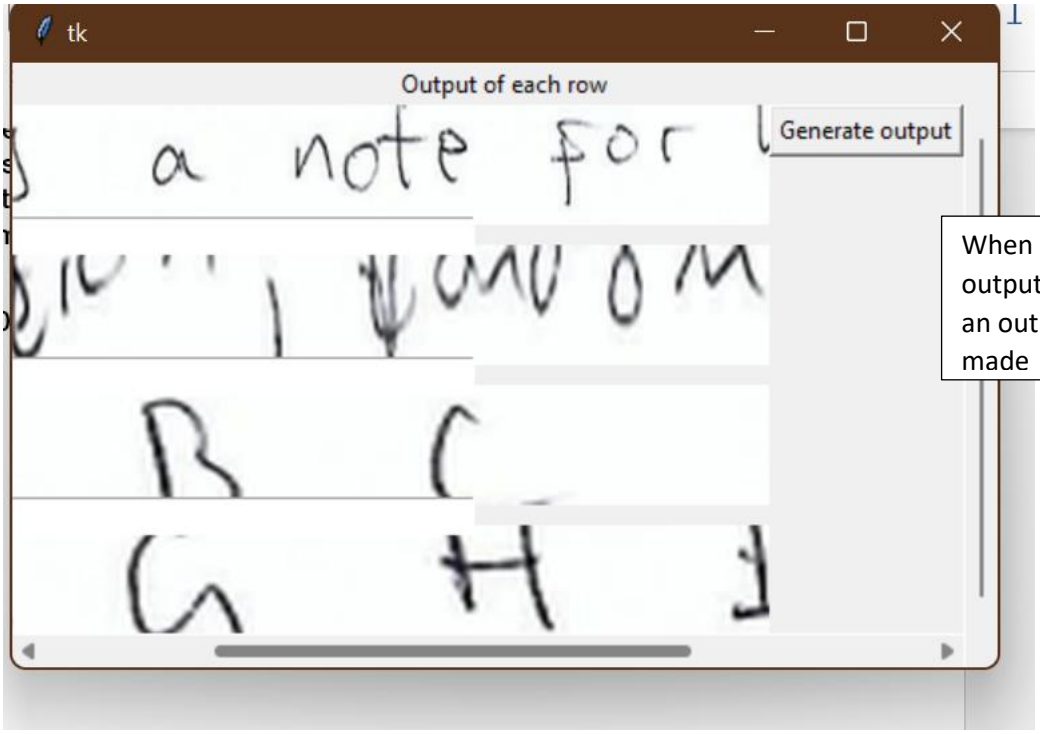
Objective 10:



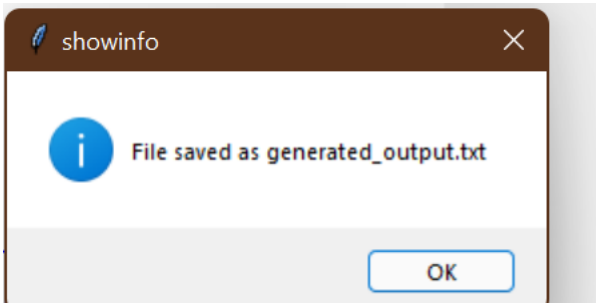
The horizontal and vertical scrollbars are very useful for my row images, because it allows the user to scroll through all the rows (if the row images exceed the canvas area) and for long rows, the user can use the horizontal scrollbar to view.

Objective 11:

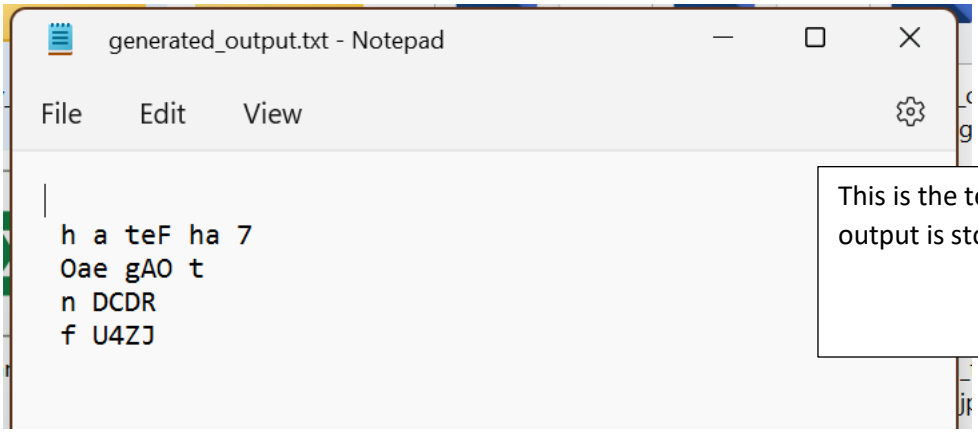
My project: Handwriting recognition research project



When “Generate output” is pressed an output file is made



A message box is shown as confirmation to user



This is the text file where output is stored

Once I click the “generate output” button, I get a text file as an output. The user can copy the contents of their text file and repurpose it into the place they want to use it in.

Section 4.4) Limitations of my program

The main limitation in my program is that my letter detection and recognition is very weak and very inconsistent. This single-handedly makes my project impractical for a real-life use, because it will produce inconsistent and highly erroneous text outputs, which is useless for the user.

Another limitation of my program is the the Tkinter implementation of the GUI. I haven't talked about this enough in this project but using Tkinter is extremely laggy and slow when you enter files with a lot of letters detected. In some cases, I had to restart my Python IDE because my program kept crashing due to high load. This is also a reason why the user experience of this program is so below par.

Section 4.5) Maintenance issues and future improvements

If I were to continue developing my project further, I would change many things. First of all, I would train my neural network (using TensorFlow) better. I would plot accuracy against epochs graphs for every parameter I change during training my model and I would find the optimal model (with the highest accuracy). Due to time-constraints of the project, and my lack of time management (because I spent too much time making my own CNN which failed), I couldn't make the optimal neural network, which has affected my program accuracy in recognising handwritten letters.

Another thing I would improve on if I am to develop my project further, is my image processing algorithms. In my post development testing, I saw that my program wasn't detecting many of the letters in the program which is a huge problem because the letters cannot be recognised, and a feasible output cannot be reached. If I had more time, I would find/develop the optimal thresholding algorithm and contour detection algorithm that can eliminate image noise while being focused on the foreground.

Last thing I would change is the GUI. If I will continue to develop this program further, I will make a web based/web API solution. This is because, websites are more portable than Python Tkinter programs. You can use a website on a computer, but also on a portabl mobile phone. Using a web based GUI can increase the scope of my project, and I can make my implementation of the solution more practical for real life.

My final code:

finalGUI.py

My project: Handwriting recognition research project

```
"""
Created on Fri Apr 15 15:49:45 2022

@author: Sahil
"""
from tkinter import *
from PIL import Image, ImageTk
import numpy as np
import cv2
import tensorflow as tf
import spacing_for_letter
import thickness_of_letter_enlarged
import TESTING_nn
import row_sorting
import laying_out_words
import detect_letters
nn = tf.keras.models.load_model("final_CNN") # loading my CNN model
name_of_file = "3_rows_roughly.png" # I will use this when I need to call functions from external python files

def open_file():
    global name_of_file
    global image
    global imageOpen
    filetypes = (
        ("PNG file", '*.png'),
        ("JPEG/JPG file", '*.jpeg *.jpg') # these are the only types of files acceptable when using filedialog.askopenfile()
        )
    from tkinter import filedialog
    name_of_file = filedialog.askopenfilename( # this is a tkinter method that opens a window and allows the user to manually search their directories to open a file
        title = "Open a file",
        initialdir = "/",
        filetypes = filetypes # I have defined this above, PNG and JPEG/JPG are the only filetypes allowed
        )
    imageOpen = Image.open(name_of_file) # opening the file path that the user has input
    image = ImageTk.PhotoImage(image = imageOpen)
    canvas_image.itemconfig(image_plot, image = image) # displays the image onto canvas_image
    canvas_image.delete("rect")

root = Tk() #defining the main window of the program
menu1 = Menu(root) # the Menu() defines a menu for me in the window
menu_dropdown = Menu(menu1, tearoff = 0)
menu_dropdown.add_command(label = "Open", command = open_file) # this is the actual button that will allow the user to input images
menu_dropdown.add_separator()
menu1.add_cascade(label = "File operations", menu = menu_dropdown) # this is the tab the user has to click on to get to the "Open" button
root.config(menu = menu1) # stating that the main window will have a menubar
canvas_image = Canvas(root) # canvas widget is defined where I will place my input image
canvas_image.config(highlightthickness=0)
canvas_image.pack(side=LEFT, expand=YES, fill=BOTH) # .pack() allocates a coordinate space for canvas_image
imageOpen = Image.open(name_of_file) # opening input image
image = ImageTk.PhotoImage(imageOpen) # PhotoImage object that will be used to output image in canvas_image
image_plot = canvas_image.create_image(0,0, anchor="nw",image=image) # .create_image() displays my input image on the canvas
vertical_scrollbar = Scrollbar(root, orient = "vertical", command = canvas_image.yview) # defining vertical scrollbar
vertical_scrollbar.pack(side = RIGHT, fill = Y)
horizontal_scrollbar = Scrollbar(root, orient = "horizontal", command = canvas_image.xview) # defining horizontal scrollbar
horizontal_scrollbar.pack(side = BOTTOM, fill = X) # placement of this scrollbar is at the bottom
horizontal_scrollbar.config(command=canvas_image.xview)
vertical_scrollbar.config(command=canvas_image.yview)
canvas_image.configure(yscrollcommand = vertical_scrollbar.set, xscrollcommand = horizontal_scrollbar.set) # this is where I assign the scrollbar command onto canvas_image specifically
canvas_image.configure(scrollregion=canvas_image.bbox("all"))
canvas_image.config(scrollregion=(0,0,np.shape(imageOpen)[1], np.shape(imageOpen)[0]))
label = Label(canvas_image)
label.pack()
scale_factor = 1
```

```
scale_factor = 1
def zoom(event):
    global image
    global scale_factor
    global counter
    scale_factor = slider1.get() # this .get() method receives the input that the user gives for Scale()
    width = int(np.shape(imageOpen)[1]*scale_factor/100) # I change the width of the original image using scale factor
    height = int(np.shape(imageOpen)[0]*scale_factor/100) # I change the height of the original image using scale factor
    if width == 0:
        width = 1
    if height == 0:
        height = 1
    imageResize = imageOpen.resize((width, height)) # resizing image based on new width and height
    image = ImageTk.PhotoImage(image = imageResize)
    canvas_image.itemconfig(image_plot, image = image) # .itemconfig is used to update imageplot which deals with displaying images (in above code)
    canvas_image.delete("rect")
    resize_bounding_with_zoom()
    if counter == 1:
        recognition()
label = Label(root, text = "Zoom: ")
label.pack()
slider1 = Scale(root, from_ = 0, to = 200, orient = HORIZONTAL, command = zoom) # this is the slider that is used for zooming, with minimum value = 0, maximum value = 200
slider1.pack()
coor_info = []
def detect():
    global coor_info # I have used coor_info as a global variable so I can use it throughout the program
    import detect_letters # importing my program detect_letters
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    import row_sorting
    output = row_sorting.main(coor_info, height)
    #print("output: " + str(output))
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0][0]*scale_factor/100)) # x coordinate of bounding box
        y_coor = int(round(coor_info[x][0][0][1]*scale_factor/100)) # y coordinate of bounding box
        x_coor_w = int(round(coor_info[x][0][1][0]*scale_factor/100)) # x + w coordinate of bounding box
        y_coor_h = int(round(coor_info[x][0][1][1]*scale_factor/100)) # # y + h coordinate of bounding box
        #print(x_coor, y_coor, x_coor_w, y_coor_h)
        canvas_image.create_rectangle(x_coor, y_coor, x_coor_w, y_coor_h, outline = "blue", tags = "rect") # creates bounding box
        #text_to_print = "x:{}, y:{}, num: {}".format(x_coor_w, y_coor_h, coor_info[x][1]) # text for each bounding box showing its num (from detect letters) and current coordinates
        #canvas_image.create_text(x_coor_w, y_coor_h, text = text_to_print, fill = "black", tags = "rect") # outputs text
def resize_bounding_with_zoom():
    global coor_info # I have used coor_info from detect as a global variable so I can use it throughout the program
    global scale_factor
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0][0]*scale_factor/100)) # x coordinate of bounding box after scale factor
        y_coor = int(round(coor_info[x][0][0][1]*scale_factor/100)) # y coordinate of bounding box after scale factor
        x_coor_w = int(round(coor_info[x][0][1][0]*scale_factor/100)) # x + w coordinate of bounding box after scale factor
        y_coor_h = int(round(coor_info[x][0][1][1]*scale_factor/100)) # # y + h coordinate of bounding box after scale factor
        #print(x_coor, y_coor, x_coor_w, y_coor_h)
        canvas_image.create_rectangle(x_coor, y_coor, x_coor_w, y_coor_h, outline = "blue", tags = "rect")
        #text_to_print = "x:{}, y:{}, num: {}".format(x_coor_w, y_coor_h, coor_info[x][1])
        #canvas_image.create_text(x_coor_w, y_coor_h, text = text_to_print, fill = "black",  tags = "rect")
detect_button = Button(root, text = "Detect", command = detect)
detect_button.pack()
def delete_canvas_rectangles():
    canvas_image.delete("rect")
delete_button = Button(root, text = "Delete", command = delete_canvas_rectangles)
delete_button.pack()
counter = 0
```

My project: Handwriting recognition research project

```
counter = 0
def recognition():
    global counter
    counter = 1
    global nn
    global scale_factor
    image_recog_list = []
    for x in range(0, len(coor_info)):
        num = coor_info[x][1] # this is the unique identifier for each letter (letter saved as "letter_image" + num)
        path = "letter_image{}.jpg".format(num) # path of letter
        #print("path of original: {}".format(path))

        img_letter = cv2.imread(path)  # image is loaded
        spacing_for_letter.main(img_letter, path) # spacing is applied on letter
        path = "spaces_{}".format(path)
        #print("path of spaces: {}".format(path))

        img_space = cv2.imread(path) # image is loaded
        thickness_of_letter_enlarged.main(img_space, path) # thickness of letter is enlarged
        path = "thick_{}".format(path)
        #print("path of thick: {}".format(path))

        image_for_pred = TESTING_nn.process_letter(path) # image is processed for tensorflow neural network
        output = nn.predict(image_for_pred) # the model.predict() takes input image and gives out an output class of size 47 containing probabilities of output
        result = np.argmax(output) # np.argmax() outputs the index of the highest probabilimty in the array
        alphabet = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','d','e','f','g','h','n','q','r','t']
        #alphabet is the mapping that maps the output index of result to a letter.
        image_recog_list.append(alphabet[result]) # the result of neural network (letter) is added to array
    for x in range(0, len(coor_info)):
        x_coor = int(round(coor_info[x][0][0][0]*scale_factor/100)) # x coordinate of bounding box after scale factor
        y_coor = int(round(coor_info[x][0][0][1]*scale_factor/100)) # y coordinate of bounding box after scale factor
        x_coor_w = int(round(coor_info[x][0][1][0]*scale_factor/100)) # x + w coordinate of bounding box after scale factor
        y_coor_h = int(round(coor_info[x][0][1][1]*scale_factor/100)) # # y + h coordinate of bounding box after scale factor
        #print(x_coor, y_coor, x_coor_w, y_coor_h)
        canvas_image.create_rectangle(x_coor, y_coor, x_coor_w, y_coor_h, outline = "blue", tags = "rect")
        #text_to_print = "{}".format(image_recog_list[x]) # result of recognition is shown
        canvas_image.create_text(x_coor_w, y_coor_h, text = text_to_print, fill = "black",  font=('Helvetica 15 bold') , tags = "rect")
def recognition_to_words():
    global name_of_file
    import detect_letters # importing my program detect_letters
    coor_info = detect_letters.main(name_of_file) # coordinate information of letters
    height = np.shape(Image.open(name_of_file))[0]
    width = np.shape(Image.open(name_of_file))[1]
    import row_sorting
    output_containing_rows = row_sorting.main(coor_info, height) # this output contains letters that are sorted out into rows
    import laying_out_words
    output_fully_structured = laying_out_words.main(output_containing_rows, width) # this output contains letters that are sorted out into rows and words in each line
    #print(output_fully_structured) # to visually observe the dimensions of output_fully_structured
    final_word_letter_list = []
    for x in range(0, len(output_fully_structured)): # x loops in output_fully_structured (number of rows)
        temp = []
        for y in range(0, len(output_fully_structured[x])): # y loops in output_fully_structured (inside each row)
            temp1 = []
            for z in range(0, len(output_fully_structured[x][y])): # z loops inside each word inside each row
                #print("x: {}, y: {}, z: {}".format(x, y, z))
                num = output_fully_structured[x][y][z][1] # this is the unique identifier for each letter (letter saved as "letter_image" + num)
                path = "letter_image{}.jpg".format(num) # path of letter
                #print("path of original: {}".format(path))
                img_letter = cv2.imread(path)  # image is loaded
                spacing_for_letter.main(img_letter, path) # spacing is applied on letter
                path = "spaces_{}".format(path)
                #print("path of spaces: {}".format(path))

                img_space = cv2.imread(path) # image is loaded
                thickness_of_letter_enlarged.main(img_space, path) # thickness of letter is enlarged
                path = "thick_{}".format(path)
                #print("path of thick: {}".format(path))
```

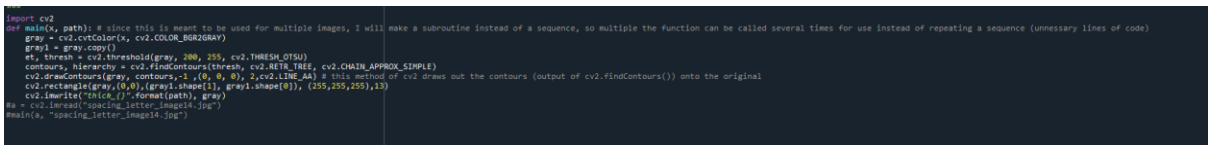
detect_letters.py

```
import cv2
import numpy as np
def main(image1):
    imageOpen = cv2.imread(image1) # read image
    width = int(imageOpen.shape[1]*200/100) # width of image is reduced by scale factor 2
    height = int(imageOpen.shape[0]*200/100) # height of image is reduced by scale factor 2
    size1 = (width,height)
    imageResize = cv2.resize(imageOpen, size1) # cv2 method resize() resizes the original image
    '''imageResize = cv2.blur(imageResize, (2, 2)) # blurs the image to try and reduce the background noise
    imageGray = cv2.cvtColor(imageResize, cv2.COLOR_BGR2GRAY) # converting image to grayscale as thresholding only accepts grayscale images
    ret, thresh = cv2.threshold(imageGray, 150, 255, cv2.THRESH_BINARY) # Binary thresholding, if pixel value is over 100, then set it to 255, otherwise 0
    contours, h = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) # this is the method of cv2 that finds contours in an image (i.e. edges)
    # the array contours contains coordinates (x,y) of each contour detected (where it starts and ends) in the image
    '''
    grayScale = cv2.cvtColor(imageResize, cv2.COLOR_BGR2GRAY)
    unwanted, binary = cv2.threshold(grayScale, 200, 255, cv2.THRESH_BINARY)
    unwanted1, invBinary = cv2.threshold(binary, 50, 255, cv2.THRESH_BINARY_INV)
    contours, hierarchy = cv2.findContours(invBinary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    num = 0
    coor_info = []
    for c in range(0, len(contours)): # I can use contour coordinates to locate letters, I will therefore use a loop to print each contour detected in the image
        x, y, w, h = cv2.boundingRect(contours[c]) # this cv2.boundingRect() outputs the x,y coordinates and the width and height of each contour, I can use this information to plot rectangles on the image where contours are.
        area_of_contour = cv2.contourArea(contours[c]) # cv2.contourArea outputs the area of the detected contour, I could have made this function myself but this method is more mathematical and gives an accurate measure of area.
        if round(area_of_contour) < 250.0: # if the area of contour is less than a certain threshold, I will count the counter as noise and therefore ignore it (next iteration of contours)
            continue
        num += 1
        crop = imageResize[y:y+h, x:x+w] # I use numpy slicing to extract the region of the image containing the contour
        name = "Letter_image{}.jpg".format(num)
        cv2.imwrite(name, crop) # saving each contour
        #rect = cv2.rectangle(imageResize, (x, y), (x + w, y + h), (0, 255, 255)) # plotting a rectangle where the contour is, using outputs of cv2.BoundingRect()
        decreasing_fac = 2
        coor_info.append([[x/decreasing_fac,y/decreasing_fac), ((x+w)/decreasing_fac, (y+h)/decreasing_fac)], num])
    #cv2.imshow("a", imageResize)
    #cv2.waitKey()
    return coor_info
```

spacing_for_letter.py

My project: Handwriting recognition research project

thickness_for_letter_enlarged.py



row_sorting.py

```
Created on Fri Apr 15 11:44:52 2022

@author: Sahil
"""
def bubbleSort_sort_row(list1): # I will perform bubble sort on my array
    while True: # while loop doesn't stop until there is no adjacency swap
        swap = False
        for x in range(0, len(list1)-1):
            if list1[x][0][1][1] > list1[x+1][0][1][1]: # if the adjacent (to the right) element is smaller than the current then swap
                swap = True
                temp = list1[x]
                list1[x] = list1[x+1]
                list1[x+1] = temp # elements swapped[4]: row_sorted = main(coor_info)
        if swap == False:
            return list1 # if there is no swaps in the list anymore, then bubble sort is complete, return new list
def main(coor_info, height): #I have made a function that takes the list of coordinates from detect_letter.py as a parameter
    y_sorted = bubbleSort_sort_row(coor_info) # first we need to bubble sort the coordinates so adjacent elements can be compared
    # note the relative position inside a line doesn't matter yet (we will rearrange that once we work out our arrays of each line)
    rows_sorted = []
    temp = 0 # I will use temp in slicing of my array to mark the first element of a new line
    for x in range(0, len(y_sorted)-1):
        if abs(y_sorted[x][0][1][1] - y_sorted[x+1][0][1][1])/height > 0.07: # if the difference between two y-coordinates (adjacent)/height > 0.07 means there is a new line
            rows_sorted.append(y_sorted[temp: x + 1]) # a new row is added with slicing between first index of the previous line (temp) and current index
            temp = x+1 # the current index becomes the new temp
        if x == len(y_sorted)-2: # I noticed during testing that the last line of the image was not saved, so this is an if statement to fix that
            rows_sorted.append(y_sorted[temp: x+2])
    return rows_sorted # return sorted list
```

laying_out_words.py

```
def bubbleSort_sort_column(list1): # I will perform bubble sort on my array
    while True: # while loop doesn't stop until there is no adjacency swap
        swap = False
        for x in range(0, len(list1)-1):
            if list1[x][0][1][0] > list1[x+1][0][1][0]: # if the adjacent (to the right) x coordinate is smaller than the current then swap
                swap = True
                temp = list1[x]
                list1[x] = list1[x+1]
                list1[x+1] = temp # elements swapped
        if swap == False:
            return list1 # if there is no swaps in the list anymore, then bubble sort is complete, return new list
def main(sorted_rows, width): #I have made a function that takes the list of coordinates from row_sorting.py as a parameter
    new_sorted_individual_row = []
    for x in range(0, len(sorted_rows)):
        a = bubbleSort_sort_column(sorted_rows[x]) # within each row, the list is sorted by x coordinates
        new_sorted_individual_row.append(a) # new list is appended to new_sorted_individual_row
    #return new_sorted_individual_row
    words_list_line_by_line = [] # this will store the rows and the word in the image
    for x in range(0, len(new_sorted_individual_row)):
        temp = 0  # I will use temp in slicing of my array to mark the first element of a new word
        word_row_list = [] # this stores the words made within each row
        for y in range(0, len(new_sorted_individual_row[x])-1):
            print("num inside Loop: {}".format(new_sorted_individual_row[x][y][1]))
            if abs(new_sorted_individual_row[x][y][0][1][0] - new_sorted_individual_row[x][y+1][0][1][0])/width > 0.08: # if the difference between two adjacent x-coordinates/width > 0.095, form a word
                print("entered inside if statement at x = {}, y = {}, num = {}".format(x,y,new_sorted_individual_row[x][y][1]))
                #print("slicing {}".format(new_sorted_individual_row[x][temp:y+1]))
                word_row_list.append(new_sorted_individual_row[x][temp:y+1]) # add the elements between temp and current index to word_row_list (new word)
                temp = y+1 # update temp to index of first letter in a new word
            if y == len(new_sorted_individual_row[x])-2: # I noticed during testing that the last line of the image was not saved, so this is an if statement to fix that
                word_row_list.append(new_sorted_individual_row[x][temp: y+2])
        words_list_line_by_line.append(word_row_list) # add each row of words to words_list_line_by_line
    return words_list_line_by_line # output
```

TESTING_nn.py

```
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image, ImageOps
#nn = tf.keras.models.load_model("final_CNN") # loading my CNN model
#from PIL import Image, ImageOps
def process_letter(img_path):
    image = Image.open(img_path) # opening an image using its path
    image_resized = image.resize((28,28)) # resizing image to 28 by 28 so that inverse and grayscale operations can be performed
    image_gray = image_resized.convert("L") # converts to grayscale
    image_inverse = ImageOps.invert(image_gray) # inverts the image so the background is black and foreground is white
    numpy_image = np.array(image_inverse) # converting to numpy array for reshaping it for CNN
    final1 = numpy_image.reshape(1, 28, 28, 1) # resizing array for CNN
    final1 = final1/255 # normalising the data so it is continuous
    return final1
```

new_neural_network_tensorflow.py

My project: Handwriting recognition research project

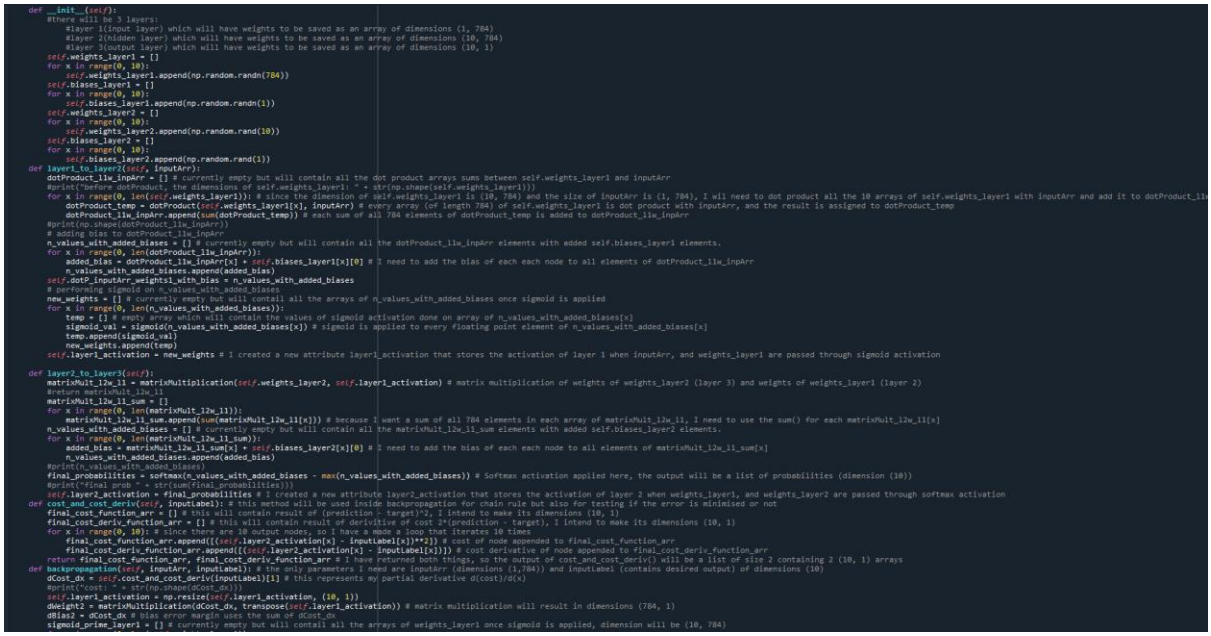
```
import pandas as pd
import numpy as np
training_data = pd.read_csv("emnist-balanced-train.csv") # reading training data from csv file using pandas module
testing_data = pd.read_csv("emnist-balanced-test.csv") # reading testing data from csv file using pandas module
#size of training_data is (88799, 785)
#size of testing_data is (14799, 785)
train_data = np.array(training_data, dtype = np.float64) # converts training_data to numpy arrays so elements can be called using its indexes
test_data = np.array(testing_data, dtype = np.float64) # converts testing_data to numpy arrays so elements can be called using its indexes
train_labels = []
for x in range(0, len(train_data)):
    train_labels.append(train_data[x][0]) # I add the first element of train_data[x] (the label) to train_labels
test_labels = []
for x in range(0, len(test_data)):
    test_labels.append(test_data[x][0]) # I add the first element of test_data[x] (the label) to test_labels
train_data = train_data/255.0
test_data = test_data/255.0
new_train_data = []
for x in range(0, len(train_data)):
    img = np.resize(train_data[x][1:], (28,28)) # I have to reshape the EMNIST data from (784) to (28, 28) for np.fliplr() and np.rot90() to work
    img_flip = np.fliplr(img) # flips image
    img_rotate = np.rot90(img_flip) # rotates image by 90 degrees
    new_train_data.append(img_rotate) # save it back as an element of train_data
new_test_data = []
for x in range(0, len(test_data)):
    img = np.resize(test_data[x][1:], (28,28)) # I have to reshape the EMNIST data from (784) to (28, 28) for np.fliplr() and np.rot90() to work
    img_flip = np.fliplr(img) # flips image
    img_rotate = np.rot90(img_flip) # rotates image by 90 degrees
    new_test_data.append(img_rotate) # save it back as an element of test_data
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32,3, input_shape=(28,28, 1)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(input_shape=(28,28, 1)),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(47,activation='softmax')
])
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
new_train_data = np.array(new_train_data, dtype = np.float64) # I will convert new_train_data to a NumPy array as tensorflow CNN can only accept NumPy arrays
new_train_data = np.resize(new_train_data, (len(new_train_data), 28, 28, 1)) # since the input for the sequential CNN is (28, 28, 1) we need to resize the new_train_data to facilitate that
train_labels = np.resize(train_labels, (len(train_labels), 1)) # the training label array is being formatted from list of number labels to list of arrays containging number labels
new_test_data = np.array(new_test_data, dtype = np.float64) # I will convert new_train_data to a NumPy array as tensorflow CNN can only accept NumPy arrays
new_test_data = np.resize(new_test_data, (len(new_test_data), 28, 28, 1)) # since the input for the sequential CNN is (28, 28, 1) we need to resize the new_test_data to facilitate that
test_labels = np.resize(test_labels, (len(test_labels), 1)) # the testing label array is being formatted from list of number labels to list of arrays containging number labels
train_labels = np.array(train_labels, np.uint8)
test_labels = np.array(test_labels, np.uint8)
history = model.fit(new_train_data, train_labels,  epochs = 50, validation_data=(new_test_data, test_labels))
model.save("final_CNN")
```

Neural_network_from_scratch.py (this is not part of my final solution but a lot of my documentation refers to my work in this file)

```
import numpy as np
from scipy.special import expit, softmax
##Miscellaneous functions##


def dotProduct(arr1, arr2): #taking two arrays (arr1 and arr2) as parameters
    if np.shape(arr1) != np.shape(arr2): # since vector dot product requires both vectors to be of the same dimensions, I have used a selection to check the dimensions of both arrays (parameters).
        return False # if the dimensions of arr1 and arr2 are not equal, then return the Boolean value False, as dot product cannot be carried out
    else:
        if len(np.shape(arr1)) == 1: # if there are no columns then I have to remove one of the iterative loops
            result = [] #assigned result to an empty array, this will be used to store the result of dot product
            for x in range(0, len(arr1)):
                temp = arr1[x] * arr2[x] # the result of multiplication between each element of arr1 and element of arr2 at corresponding position is added to temp
                result.append(temp)
            return result
        result = [] #assigned result to an empty array, this will be used to store the result of dot product
        for x in range(0, len(arr1)):
            temp = [] # stores dot product results for each arr1[x] and arr2[x]
            for y in range(0, len(arr1[x])):
                temp.append(arr1[x][y] * arr2[x][y]) # the result of multiplication between each element of arr1[x] and element of arr2[x] at corresponding position is added to temp
            result.append(temp) #at the end of each iteration of x, array temp is appended to result
        return result # at the end of dot product between arr1 and arr2, the result of the dot product is returned.
def sigmoid(x): #function sigmoid() takes x as a parameter where x is a floating point number
    return expit(x) #since sigmoid is 1/(1+e^(-x)), I have used np.exp() for euler's number, where np.exp() takes x as a parameter.
def sigmoid_prime(x): #function sigmoid_prime() takes x as a parameter where x is a floating point number
    return expit(x)*(1-expit(x)) #since sigmoid is 1/(1+e^(-x)), I have used np.exp() for euler's number, where np.exp() takes x as a parameter.

def softmax1(x): # the parameter x will be an array with dimensions (10, 784) whic represents layer 2 nodes
    exp_total = 0 # this will contain the sum of every element of every array of x, which will be used as a denominator for the softmax formula
    for index in range(0, len(x)): # iteration in every node of layer 2 (i.e. x)
        exp_total += round(np.exp(x[index]), 6) # I will increment (x[index]) to exp_total, raised as a power of euler's number
    exp_final_prob = [] # this will store the final probabilities of each node in relation with every node in layer 2
    for index in range(0, len(x)): # iterates for every index between x
        element = round(np.exp(x[index])/exp_total, 6) # this is where the softmax formula is applied
        exp_final_prob.append(element)
    return exp_final_prob # final list of probabilities
def matrixMultiplication(mat1, mat2):
    counter_mat1 = False # if mat1 has no column, the counter is set true so the dimensions can be temporarily changed before being restored
    counter_mat2 = False# if mat2 has no column, the counter is set true so the dimensions can be temporarily changed before being restored
    row_mat1 = len(mat1)
    row_mat2 = len(mat2)
    #print(len(np.shape(mat1)))
    #print(len(np.shape(mat2)))
    if len(np.shape(mat1)) == 1: # case for if mat has no columns, then len(mat[0]) is an error, so I am validating that
        a = []
```

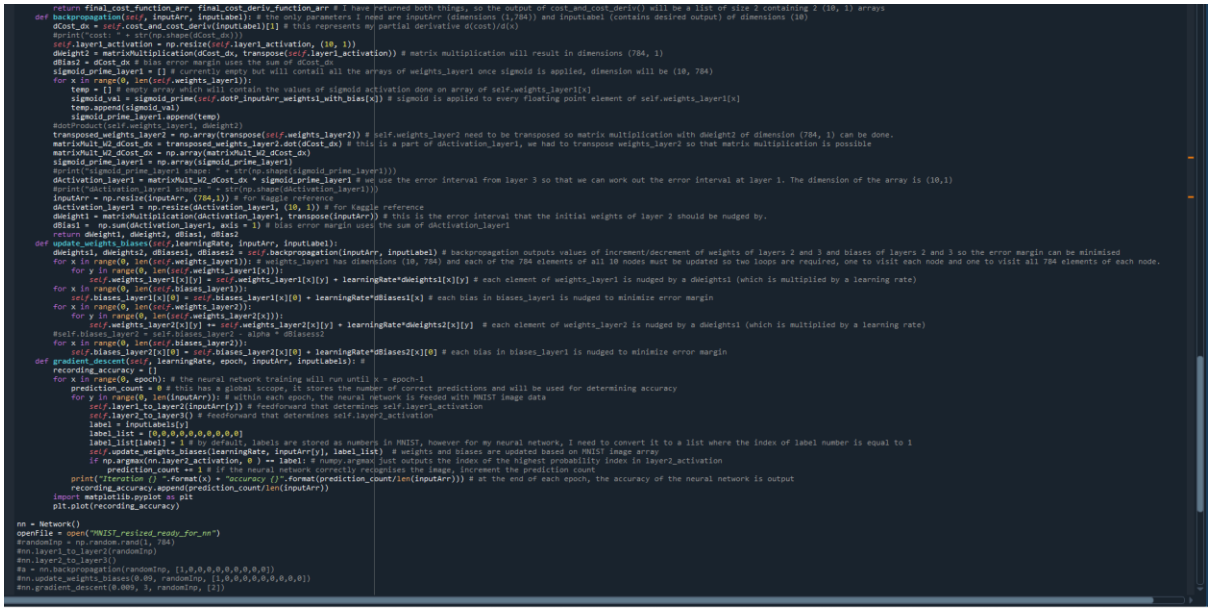
My project: Handwriting recognition research project

```
    #print(len(np.shape(mat2)))
    if len(np.shape(mat1)) == 1: # case for if mat has no columns, then len(mat[0]) is an error, so I am validating that
        a = []
        for x in range(0, len(mat1)): # I am changing the dimensions of mat1 so the dimensions have a column in it, this way my matrix multiplication will work
            a.append([mat1[x]]) # adding [mat[x]] will add an extra dimension to the array
        mat1 = a # assigning new array to mat1
        counter_mat1 = True
    column_mat1 = len(mat1[0])
    if len(np.shape(mat2)) == 1:
        a = []
        for x in range(0, len(mat2)): # I am changing the dimensions of mat1 so the dimensions have a column in it, this way my matrix multiplication will work
            a.append([mat2[x]]) # adding [mat[x]] will add an extra dimension to the array
        mat2 = a # assigning new array to mat2
        counter_mat2 = True
    column_mat2 = len(mat2[0])
    if column_mat1 != row_mat2:
        return False
    else:
        final_dim = (row_mat1, column_mat2)
        final_list = []
        for x in range(0, row_mat1):
            row = []
            for y in range(0, column_mat2):
                row.append(0)
            final_list.append(row)
        for x in range(0, row_mat1):
            for y in range(0, column_mat2):
                total = 0
                for z in range(0, column_mat1):
                    total += mat1[x][z] * mat2[z][y]
                final_list[x][y] = total
        if counter_mat1 == True or counter_mat2 == True: # if initially mat1 or mat2 had no columns, then the resulting matrix won't have columns, we need to restore the final matrix
            final_list = np.resize(final_list, np.shape(final_list)[0]) # resized so dimensions only have (np.shape(final_list)[0]) and not columns
        return final_list
def transpose(arr1): # for backpropagation, certain matrix multiplication calculations wouldn= not work unless you transpose them, so this is my implementation
    counter_mat1 = False # if mat1 has no column, the counter is set true so the dimensions can be temporarily changed before being restored
    if len(np.shape(arr1)) == 1: # case for if mat has no columns, then len(mat[0]) is an error, so I am validating that
        return arr1
    final_list = [] # currently empty but will contain results of transposing
    for x in range(0, len(arr1[0])): # I will iterate in the number of column first so I can locate element at every arr1 row at a given column and put them in a new list
        temp = [] # will contain elements from every row for every given column(x)
        for y in range(0, len(arr1)):
            temp.append(arr1[y][x]) # elements from every row for every given row(x) is appended to temp
        final_list.append(temp) # the new row is added to final_list()
    return final_list # the results are outputted
class Network:
    def __init__(self):
        #there will be 3 layers:
            #layer 1(input layer) which will have weights to be saved as an array of dimensions (1, 784)
```

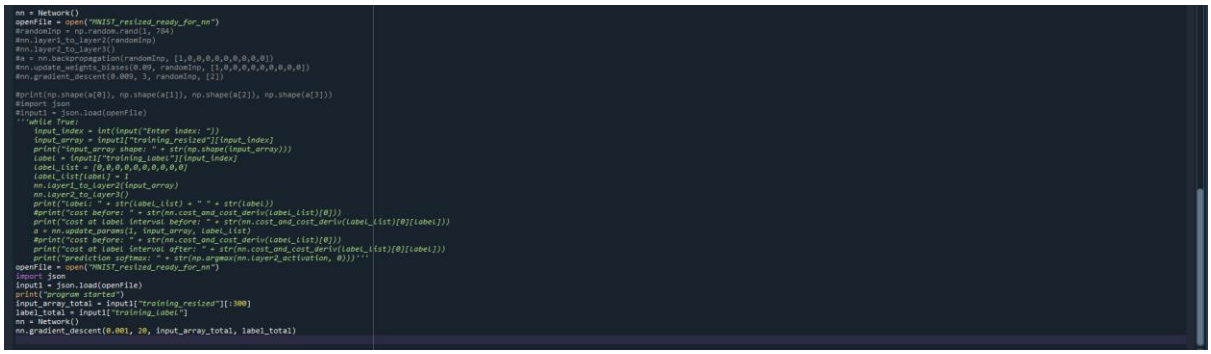


```
    def __init__(self):
        #there will be 3 layers:
            #layer 1(input layer) which will have weights to be saved as an array of dimensions (1, 784)
            #layer 2(hidden layer) which will have weights to be saved as an array of dimensions (10, 784)
            #layer 3(output layer) which will have weights to be saved as an array of dimensions (10, 1)
        self.weights_layer1 = []
        for x in range(0, 10):
            self.weights_layer1.append(np.random.randn(784))
        self.biases_layer1 = []
        for x in range(0, 10):
            self.biases_layer1.append(np.random.randn(1))
        self.weights_layer2 = []
        for x in range(0, 10):
            self.weights_layer2.append(np.random.rand(10))
        self.biases_layer2 = []
        for x in range(0, 10):
            self.biases_layer2.append(np.random.rand(1))
    def layer1_to_layer2(self, inputArr):
        dotProduct_l1w_inpArr = [] # currently empty but will contain all the dot product arrays sums between self.weights_layer1 and inputArr
        #print("before dotProduct, the dimensions of self.weights_layer1: " + str(np.shape(self.weights_layer1)))
        for x in range(0, len(self.weights_layer1)): # since the dimension of self.weights_layer1 is (10, 784) and the size of inputArr is (1, 784), I will need to dot product all the 10 arrays of self.weights_layer1 with inputArr and add it to dotProduct_l1w
            dotProduct_temp = dotProduct(self.weights_layer1[x], inputArr) # every array (of length 784) of self.weights_layer1 is dot product with inputArr, and the result is assigned to dotProduct_temp
            dotProduct_l1w_inpArr.append(sum(dotProduct_temp)) # each sum of all 784 elements of dotProduct_temp is added to dotProduct_l1w_inpArr
        #print(np.shape(dotProduct_l1w_inpArr))
        # adding bias to dotProduct_l1w_inpArr
        n_values_with_added_biases = [] # currently empty but will contain all the dotProduct_l1w_inpArr elements with added self.biases_layer1 elements.
        for x in range(0, len(dotProduct_l1w_inpArr)):
            added_bias = dotProduct_l1w_inpArr[x] + self.biases_layer1[x][0] # I need to add the bias of each each node to all elements of dotProduct_l1w_inpArr
            n_values_with_added_biases.append(added_bias)
        self.dotP_inputArr_weights1_with_bias = n_values_with_added_bieses
        # performing sigmoid on n_values_with_added_biases
        new_weights = [] # currently empty but will contail all the arrays of n_values_with_added_biases once sigmoid is applied
        for x in range(0, len(n_values_with_added_biases)):
            temp = [] # empty array which will contain the values of sigmoid activation done on array of n_values_with_added_biases[x]
            sigmoid_val = sigmoid(n_values_with_added_biases[x]) # sigmoid is applied to every floating point element of n_values_with_added_biases[x]
            temp.append(sigmoid_val)
            new_weights.append(temp)
        self.layer1_activation = new_weights # I created a new attribute layer1_activation that stores the activation of layer 1 when inputArr, and weights_layer1 are passed through sigmoid activation

    def layer2_to_layer3(self):
        matrixMult_l2w_l1 = matrixMultiplication(self.weights_layer2, self.layer1_activation) # matrix multiplication of weights of weights_layer2 (layer 3) and weights of weights_layer1 (layer 2)
        #return matrixMult_l2w_l1
        matrixMult_l2w_l1_sum = []
        for x in range(0, len(matrixMult_l2w_l1)):
            matrixMult_l2w_l1_sum.append(sum(matrixMult_l2w_l1[x])) # because I want a sum of all 784 elements in each array of matrixMult_l2w_l1, I need to use the sum() for each matrixMult_l2w_l1[x]
        n_values_with_added_biases = [] # currently empty but will contain all the matrixMult_l2w_l1_sum elements with added self.biases_layer2 elements.
        for x in range(0, len(matrixMult_l2w_l1_sum)):
            added_bias = matrixMult_l2w_l1_sum[x] + self.biases_layer2[x][0] # I need to add the bias of each each node to all elements of matrixMult_l2w_l1_sum[x]
            n_values_with_added_biases.append(added_bias)
        #print(n_values_with_added_biases)
        final_probabilities = softmax(n_values_with_added_biases - max(n_values_with_added_biases)) # Softmax activation applied here, the output will be a list of probabilities (dimension (10))
        #print("final prob " + str(sum(final_probabilities)))
        self.layer2_activation = final_probabilities # I created a new attribute layer2_activation that stores the activation of layer 2 when weights_layer1, and weights_layer2 are passed through softmax activation
    def cost_and_cost_deriv(self, inputLabel): # this method will be used inside backpropagation for chain rule but also for testing if the error is minimised or not
        final_cost_function_arr = [] # this will contain result of (prediction - target)^2, I intend to make its dimensions (10, 1)
        final_cost_deriv_function_arr = [] # this will contain result of derivitive of cost 2*(prediction - target), I intend to make its dimensions (10, 1)
        for x in range(0, 10): # since there are 10 output nodes, so I have a made a loop that iterates 10 times
            final_cost_function_arr.append([(self.layer2_activation[x] - inputLabel[x])**2]) # cost of node appended to final_cost_function_arr
            final_cost_deriv_function_arr.append([(self.layer2_activation[x] - inputLabel[x])]) # cost derivative of node appended to final_cost_deriv_function_arr
        return final_cost_function_arr, final_cost_deriv_function_arr # I have returned both things, so the output of cost_and_cost_deriv() will be a list of size 2 containing 2 (10, 1) arrays
    def backpropagation(self, inputArr, inputLabel): # the only parameters I need are inputArr (dimensions (1,784)) and inputLabel (contains desired output) of dimensions (10)
        dCost_dx = self.cost_and_cost_deriv(inputLabel)[1] # this represents my partial derivative d(cost)/d(x)
        #print("cost: " + str(np.shape(dCost_dx)))
        self.layer1_activation = np.resize(self.layer1_activation, (10, 1))
        dWeight2 = matrixMultiplication(dCost_dx, transpose(self.layer1_activation)) # matrix multiplication will result in dimensions (784, 1)
        dBias2 = dCost_dx # bias error margin uses the sum of dCost_dx
        sigmoid_prime_layer1 = [] # currently empty but will contail all the arrays of weights_layer1 once sigmoid is applied, dimension will be (10, 784)
```

My project: Handwriting recognition research project



```
        return final_cost_function_arr, final_cost_deriv_function_arr # I have returned both things, so the output of cost_and_cost_deriv() will be a list of size 2 (containing 2 (10, 1) arrays
    def backpropagation(self, inputArr, inputLabel): # the only parameters I need are inputArr (dimensions (1,784)) and inputLabel (contains desired output) of dimensions (10)
        dCost_dx = self.cost_and_cost_deriv(inputLabel)[1] # this represents my partial derivative d(cost)/d(x)
        #print("cost: " + str(np.shape(dCost_dx)))
        self.layer1_activation = np.resize(self.layer1_activation, (10, 1))
        dweight2 = matrixMultiplication(dCost_dx, transpose(self.layer1_activation)) # matrix multiplication will result in dimensions (784, 1)
        dBias2 = dCost_dx # bias error margin uses the sum of dCost_dx
        sigmoid_prime_layer1 = [] # currently empty but will contain all the arrays of weights_layer1 once sigmoid is applied, dimension will be (10, 784)
        for x in range(0, len(self.weights_layer1)):
            temp = [] # empty array which will contain the values of sigmoid activation done on array of self.weights_layer1[x]
            sigmoid_val = sigmoid_prime(self.dotP_inputArr_weights1_with_bias[x]) # sigmoid is applied to every floating point element of self.weights_layer1[x]
            temp.append(sigmoid_val)
            sigmoid_prime_layer1.append(temp)
        #dotProduct(self.weights_layer1, dweight2)
        transposed_weights_layer2 = np.array(transpose(self.weights_layer2)) # self.weights_layer2 need to be transposed so matrix multiplication with dweight2 of dimension (784, 1) can be done.
        matrixMult_W2_dCost_dx = transposed_weights_layer2.dot(dCost_dx) # this is a part of dActivation_layer1, we had to transpose weights_layer2 so that matrix multiplication is possible
        matrixMult_W2_dCost_dx = np.array(matrixMult_W2_dCost_dx)
        sigmoid_prime_layer1 = np.array(sigmoid_prime_layer1)
        #print("sigmoid_prime_layer1 shape: " + str(np.shape(sigmoid_prime_layer1)))
        dActivation_layer1 = matrixMult_W2_dCost_dx * sigmoid_prime_layer1 # we use the error interval from layer 3 so that we can work out the error interval at layer 1. The dimension of the array is (10,1)
        #print("dActivation_layer1 shape: " + str(np.shape(dActivation_layer1)))
        inputArr = np.resize(inputArr, (784,1)) # for Kaggle reference
        dActivation_layer1 = np.resize(dActivation_layer1, (10, 1)) # for Kaggle reference
        dweight1 = matrixMultiplication(dActivation_layer1, transpose(inputArr)) # this is the error interval that the initial weights of layer 2 should be nudged by.
        dBias1 =  np.sum(dActivation_layer1, axis = 1) # bias error margin uses the sum of dActivation_layer1
        return dweight1, dweight2, dBias1, dBias2
    def update_weights_biases(self,learningRate, inputArr, inputLabel):
        dweights1, dWeights2, dBiases1, dBiases2 = self.backpropagation(inputArr, inputLabel) # backpropagation outputs values of increment/decrement of weights of layers 2 and 3 and biases of layers 2 and 3 so the error margin can be minimised
        for x in range(0, len(self.weights_layer1)): # weights_layer1 has dimensions (10, 784) and each of the 784 elements of all 10 nodes must be updated so two loops are required, one to visit each node and one to visit all 784 elements of each node.
            for y in range(0, len(self.weights_layer1[x])):
                self.weights_layer1[x][y] = self.weights_layer1[x][y] + learningRate*dweights1[x][y] # each element of weights_layer1 is nudged by a dWeights1 (which is multiplied by a learning rate)
        for x in range(0, len(self.biases_layer1)):
            self.biases_layer1[x][0] = self.biases_layer1[x][0] + learningRate*dBiases1[x] # each bias in biases_layer1 is nudged to minimize error margin
        for x in range(0, len(self.weights_layer2)):
            for y in range(0, len(self.weights_layer2[x])):
                self.weights_layer2[x][y] += self.weights_layer2[x][y] + learningRate*dWeights2[x][y]  # each element of weights_layer2 is nudged by a dWeights1 (which is multiplied by a learning rate)
        #self.biases_layer2 = self.biases_layer2 - alpha * dBiasess2
        for x in range(0, len(self.biases_layer2)):
            self.biases_layer2[x][0] = self.biases_layer2[x][0] + learningRate*dBiases2[x][0] # each bias in biases_layer1 is nudged to minimize error margin
    def gradient_descent(self, learningRate, epoch, inputArr, inputLabels): #
        recording_accuracy = []
        for x in range(0, epoch): # the neural network training will run until x = epoch-1
            prediction_count = 0 # this has a global accope, it stores the number of correct predictions and will be used for determining accuracy
            for y in range(0, len(inputArr)): # within each epoch, the neural network is feeded with MNIST image data
                self.layer1_to_layer2(inputArr[y]) # feedforward that determines self.layer1_activation
                self.layer2_to_layer3() # feedforward that determines self.layer2_activation
                label = inputLabels[y]
                label_list = [0,0,0,0,0,0,0,0,0,0]
                label_list[label] = 1 # by default, labels are stored as numbers in MNIST, however for my neural network, I need to convert it to a list where the index of label number is equal to 1
                self.update_weights_biases(learningRate, inputArr[y], label_list)  # weights and biases are updated based on MNIST image array
                if np.argmax(nn.layer2_activation, 0 ) == label: # numpy.argmax just outputs the index of the highest probability index in layer2_activation
                    prediction_count += 1 # if the neural network correctly recognises the image, increment the prediction count
            print("Iteration {} ".format(x) + "Accuracy {}".format(prediction_count/len(inputArr))) # at the end of each epoch, the accuracy of the neural network is output
            recording_accuracy.append(prediction_count/len(inputArr))
        import matplotlib.pyplot as plt
        plt.plot(recording_accuracy)

nn = Network()
openFile = open("MNIST_resized_ready_for_nn")
#randomInp = np.random.rand(1, 784)
#nn.layer1_to_layer2(randomInp)
#nn.layer2_to_layer3()
#a = nn.backpropagation(randomInp, [1,0,0,0,0,0,0,0,0,0])
#nn.update_weights_biases(0.09, randomInp, [1,0,0,0,0,0,0,0,0,0])
#nn.gradient_descent(0.009, 3, randomInp, [2])
```



```
nn = Network()
openFile = open("MNIST_resized_ready_for_nn")
#randomInp = np.random.rand(1, 784)
#nn.layer1_to_layer2(randomInp)
#nn.layer2_to_layer3()
#a = nn.backpropagation(randomInp, [1,0,0,0,0,0,0,0,0,0])
#nn.update_weights_biases(0.09, randomInp, [1,0,0,0,0,0,0,0,0,0])
#nn.gradient_descent(0.009, 3, randomInp, [2])

#print(np.shape(a[0]), np.shape(a[1]), np.shape(a[2]), np.shape(a[3]))
#import json
#input1 = json.load(openFile)
'''while True:
    input_index = int(input("Enter index: "))
    input_array = input1["training_resized"][input_index]
    print("input_array shape: " + str(np.shape(input_array)))
    label = input1["training_label"][input_index]
    label_list = [0,0,0,0,0,0,0,0,0,0]
    label_list[label] = 1
    nn.layer1_to_layer2(input_array)
    nn.layer2_to_layer3()
    print("Label: " + str(label_list) + " " + str(label))
    #print("cost before: " + str(nn.cost_and_cost_deriv(label_list)[0]))
    print("cost at label interval before: " + str(nn.cost_and_cost_deriv(label_list)[0][label]))
    a = nn.update_params(1, input_array, label_list)
    #print("cost before: " + str(nn.cost_and_cost_deriv(label_list)[0]))
    print("cost at label interval after: " + str(nn.cost_and_cost_deriv(label_list)[0][label]))
    print("prediction softmax: " + str(np.argmax(nn.layer2_act(oution, 0))))'''
openFile = open("MNIST_resized_ready_for_nn")
import json
input1 = json.load(openFile)
print("program started")
input_array_total = input1["training_resized"][:300]
label_total = input1["training_label"]
nn = Network()
nn.gradient_descent(0.001, 20, input_array_total, label_total)
```